A company's network: automated network deployment and testing system

Ana Barros
up201806593
FEUP

João Costa
up201806560
FEUP

João Martins
up201806436
FEUP

Rui Pinto
up201806441
FEUP

***Abstract*—This paper describes the development and management of a network for a company with two geographically separated facilities. There is a detailed description of the network topology designed by the group, along with the custom template system and the deployment and testing pipeline. The approach mentioned includes references to various topics related to DevOps, including automatization, programming techniques, networks, services, and systems management.**

I. Introduction

In this project, we wanted to simulate a company that develops a web application. This company would have both web developers (working on the web app) and network/system administrators (managing the company's infrastructure).

In order to make this scenario more interesting/challenging, we decided that the workplace of these two groups of people would be geographically separated. The objective is to create a functioning network design that suits the company's needs, provides a variety of services to those working there, and is accessible via VPN, so collaborators can work remotely.

The project files are available on GitHub on the project's repository.

II. The Network Topology

Figure 1 depicts the diagram of the network. The top facility is where web application development takes place. The bottom part focuses on the management of the company's network and systems. From here on out, the top part will be referred to as **Lisbon** and the bottom part will be referred to as **Porto**.

A. Services

In Lisbon, the company serves both a web application and a database. The website is load balanced and accessible to everyone inside or outside the company's private network. It is important to allow employees without physical access to the internal network to connect through a VPN: for example, when they are at home. On top of this, there is also a DNS server and a Nagios server (for monitoring purposes) in this facility.

B. Networks and subnetworks

Regarding networks, on the Lisbon part, there is a *load_balancers* subnetwork that contains the web application's load balancers. The *internal_B* subnetwork contains the employees of the company (Employee1, Employee2, Employee3), the DHCP server (DHCP_B), the database (db) and the database managing interface (pgAdmin). The *dmz_B* network has the website, the DNS server, the VPN server (VPN_B) and the firewall (firewall_B).

Porto has two networks: *dmz_C* and *internal_C*. The *dmz_C* has the firewall (firewall_C) and the *internal_C* network has the DHCP server, the Nagios client and the company's employees (Netmanager1, Netmanager2, Netmanager3). The *pubnet* bridge connects the two parts to the internet.

There is also the *internet* bridge for Porto and Lisbon to communicate with each other. The reasons why this last bridge was needed is described in the section VII.

C. Routers

There are four routers in total. *R_C* and *R_B* are responsible for connecting the DMZ network with the internal network. The edge routers, *ER_C* and *ER_B*, are at the boundary of the company's DMZs and the rest of the internet.

III. Deployment environment

The network is deployed in 3 Virtual Machines: one for configuration/deployment, and one for each geographic location (Porto and Lisbon). This arrangement can be seen in image 2.

With this, the two parts of the company run separately in two virtual machines inside the same computer. The machine representing Porto is referred to as VMC. The machine representing Lisbon is referred to as VMB. This computer is connected to a physical switch where we have created 3 bridges: *br-lisboa* for Lisbon's internal network, *br-porto* for Porto's internal network, and *br-internet* for the network connecting the edge routers.

The configuration VM has internet access and is able to establish an SSH connection to both VMB and VMC (management network). During the provisioning process, VMB and VMC access the internet through the configuration VM. *Proxmox* [1] manages the virtualization system.

IV. Network source of truth

For this project, we created and maintained a network source of truth, which is used in the pipeline described in section VI. For each part of the company, there are two files that contain the information necessary to deploy the network: the *network architecture* file, and the *docker compose* file.

[1]Proxmox - Powerful open-source server solutions on Proxmox's Website

A. The network architecture file

The *network architecture* file is a JSON file that has information on the existing networks, and services (docker containers). An example of the declaration of a network includes the indication of its name, network address, and gateway:

Source Code 1: Network declaration example

```
{
    "name": "dmz_b",
    "subnet": "172.0.1.0/28",
    "gateway": "172.0.1.13"
},
```

The information about a service includes its name, the networks it should be connected to (along with its IP address in those networks), and the scripts (and corresponding command-line arguments) to run at startup:

Source Code 2: Service declaration example

```
{
    "name": "dhcp_b",
    "networks": [
      {
        "name": "internal_b",
        "ip": "10.0.2.4"
      }
    ],
    "entrypoints": [
      {
        "path": "./entry.sh",
        "commands": ["10.0.2.", "dns_dmz_b"]
      }
    ]
},
```

The services are docker containers. As such, each service has an entry-point: a script/program that is run when a container is started. For convention, the group decided that all services' entry-point would be a script called **entry.sh**. With this, we can wrap the *function* to be provided by each service with other common scripts, for example, setting the default gateway.

B. The docker compose file

The group uses a docker compose file to declare the different services to be deployed in a machine. This file will contain template strings that will be resolved at deploy-time based on the *network configuration* file.

Source Code 3: Network docker-compose example

```
dmz_b:
    name: dmz_b
    driver: bridge
    ipam:
      config:
        - subnet: dmz_b_subnet
          gateway: dmz_b_gateway
firewall:
    build: ./firewall
    container_name: firewall
    cap_add:
        - NET_ADMIN
    entrypoint: firewall_entrypoint
    networks:
        dmz_b:
            ipv4_address: firewall_dmz_b
```

Section VI describes the template-resolution process in greater detail.

V. Tests

By the end of every deployment, we test whether everything is working as expected. This section describes and enumerates the unit-tests that are run automatically after deployment, and other tests that can be done manually to verify some services.

A. Deployment tests

Deployment tests include:

- Web application is accessible by internal clients;
- DHCP servers are assigning valid addresses;
- The database-management service (*pgAdmin*) is accessible by internal clients;
- Clients on the internal network can access the internet;
- External hosts can access the web application;
- Nagios interface is accessible by the network managers;
- Nagios service reports the web application as healthy;
- Firewalls are up and blocking intended requests;
- Internal hosts can access the web application and the database by an internal name (served by the company's DNS).

B. Other tests

We are able to connect to the internal network of each region by connecting to the corresponding bridge on the physical switch. This was useful to test the DHCP servers in early versions of the project.

Although it would be possible to automate the connection to the VPN service, the connection to the VPN is established manually. This is because connecting to the VPN implies sharing some secrets that are generated on VPN service start up.

VI. Pipeline

The group created a pipeline to automate the setup, deployment and testing processes of each network/machine. The pipeline runs on the configuration machine.

Example running full pipeline for VMB:

```
./setup-lisboa.sh
ssh vmb 'bash -s' <lisboa_machine/deploy.sh
ssh vmb 'bash -s' <lisboa_machine/test.sh
```

A. Setup phase

The setup of the machine is done by running a setup bash script. This script starts by configuring the machines' network and installing docker (if needed). The docker install process can fail because of Ubuntu's automatic updates. As such, there is a part of the setup that is only run once that disables automatic updates and restarts the machine (to apply the change).

Afterward, it sets up a *Systemd* service file for the deployment phase, and copies the docker images needed to that machine. Finally, the templates on the docker compose file are resolved, and the result is sent to the target machine IV.

B. Deployment phase

The deployment of the network is the simplest phase of the pipeline. First, it sets up the physical network interfaces needed by the network. Then, it uses the aforementioned *Systemd* service to stop and clean-up artifacts from old runs, build the docker images (if needed), and execute *docker compose* on the template-generated file.

If no error is reported, the network and its services are now running.

C. Testing phase

This last phase runs the automatic tests described in section V. After all tests are finished, a report is generated containing information about the success state of the test (success/failure) and their output.

D. Re-deployment consequences

It should be noted that, every time services, or their configurations, or the network topology changes, the pipeline needs to be run starting from the *setup phase*. This fact, coupled with the usage of docker compose, means that any change leads to the restart of all services, which in turn leads to the loss of run-time data of these services/container images.

VII. Implementation details

This section provides insight into how some services were set up, what problems arose, and the solutions employed for those problems.

A. Porto and Lisbon Connection

In section IV, it is mentioned that there is an *internet* bridge for Porto and Lisbon to communicate with each other. Without this bridge, upon testing whether Porto could reach Lisbon (and *vice-versa*) using *ping*, this would result in a *time to live exceed* error. The reason for this error was due to the fact that the ICMP packets would eventually reach a physical router, and, from there, forwarded to the wrong interface since the router didn't know about the DMZs of both networks (didn't have the routes, so it used the default gateway).

B. DNS

Mapping domain names to addresses is a critical component of any web business or internal entrepreneurial network. It allows for better usability and robustness, as network users only need to know the names for the services/machines. Furthermore, it allows for a more malleable experience for network administrators/managers, because the IP addresses attributed may be changed without affecting the general user experience.

For this purpose, a DNS server is located in Lisbon that resolves requests from inside and outside the network. Split DNS is configured in order to serve the *pgAdmin* and *database* names only to internal users. The names that are resolved are:

- *www.mynet.org* - Gives access to the company's website, which is accessible from both inside the network and outside. As such, the DNS server needs to handle multiple requests diligently and reliably.
- *database.mynet.org* - Maps to the internal database. Only internal users from Lisbon can query for this namespace.
- *pgadmin.mynet.org* - Maps to the pg admin interface. Similar to the database namespace, only internal Lisbon users can resolve this address.

If the number of DNS requests starts to increase by large numbers, either due to an internal network expansion or due to an increase in external website visits, the company may benefit from employing secondary (caching) DNS servers to ease this load.

C. VPN

There's a VPN server on the Lisbon facility. This server runs WireGuard [1]. It is configured to accept up to 10 clients simultaneously and allows communication between those. This means that it generates 10 secrets, 1 for each peer, on the first start. Furthermore, it allows access to Lisbon's internal network to connected peers.

Our *client* docker containers have **WireGuard tools installed**. As such, these are able to connect to the VPN, using *wg-quick*, provided one of the secrets generated by the VPN server. These secrets contain information on the VPN's network and the key to use.

D. DHCP

The two DHCP servers run Internet Consortium's DHCP server implementation, DHCPD[4]. These servers are configured to serve an IP (inside a range), the network's default gateway, and the company's DNS.

In order to automatically test the DHCP servers, the containers for internal clients come with DHCPCD. This client is run in the background on this container's images, and is configured to fetch IP addresses, network gateways, and domain name servers. Running this client in docker brings two challenges/problems.

The first problem is that for the container images to be part of the network where the DHCP server is running, they need an IP address assigned. If we don't assign one, docker will assign one automatically. This problem is easily solved by

either ignoring it, or by deleting the IP before/after obtaining the lease from the DHCP server. Since DHCPCD[3] ignores the static IP (not part of a lease), it will try to obtain a new one either way.

The second problem is that DHCPCD won't be able to set the information about the new DNS automatically. When DHCPCD obtains a lease containing DNS, it relies on *resolvconf* to deal with those. Although these container images have *resolvconf* available, it isn't able to perform its function correctly, because the file */etc/resolv.conf* is a protected file inside docker contexts: the container can change its contents, but now the file. This interferes with *resolvconf's* normal working module: "hijack" */etc/resolv.conf* by symlinking it to a different file. In order to circumvent this problem, the group hooks on DHCPCD and waits for it to obtain a lease. After a lease is obtained, *resolvconf* is force-updated manually (to ensure the latest info is available) and the contents of its information file are copied to */etc/resolv.conf*. This allows us to serve DNS from the DHCP servers and have them available to clients.

E. Nagios

It is always important to monitor the health and performance of network services. In order to monitor the web application, the group uses Nagios [2]. The web application service also runs an instance of Nagios server. This server collects information about the state of the machine and application, such as: HTTP response time, disk usage, and available memory. This server only serves requests originating from the internal networks: it is not available to the general public.

In Porto's internal network, there is a service running Nagios client. This service communicates with Nagios servers (in this case only 1), and aggregates the reported data. The client is only accessible by the network admins in Porto (internal network). As the Nagios client is located in a *macvlan* network, there is no easy way of mapping the container's internal port to the host VM. Therefore, we installed a text-based web browser (*w3m*) on every network manager to allow interaction with the Nagios client web interface using the terminal.

F. Firewall

To secure the networks, we implemented a firewall in each one. There are *iptables* rules to make sure that inbound connections target the DMZ, and not the internal network. As an example, the group decided to only allow established/related connections to the internal network but not new connections, as this might turn the internal network vulnerable to unwanted connections. Furthermore, we allowed all types of connections to the DMZ network and all outgoing traffic coming from the internal and DMZ networks. Everything beyond this scope will get rejected by the firewall.

Since all incoming and outgoing traffic passes through the firewall, the firewall can be seen as a Man-in-the-Middle agent. This presents several benefits for situations where we want to inspect all the traffic flowing in and out of the network.

VIII. CI/CD

The group used *git* as the version control system for the project development. The fact that the whole deployment and testing is fully automate in a pipeline, combined with the usage of version control, means that we are able to go back to project versions and deploy them with ease. Although, it would be possible and desirable to leverage this for usage in a CI/CD pipeline, the group didn't have convenient access to one.

IX. Future Work

The group identified several points where future work should be focused. With regard to the VPN server, it would be desirable to pre-generate the VPN secrets, so peers could connect automatically. This would allow for unit-tests for the VPN service.

The pipeline implemented for this project could be improved/simplified by using an existing *templating engine*, for example, Jinja2. Coupling this with a configuration management/deployment tool like Ansible would prove useful, and increase the robustness of the system. The pipeline could also be made more modular by only restarting/re-deploying services that changed. As mentioned previously, when making a change to the services, their configuration, or the network topology, all services need to be restarted. By using a tool like *terraform* or *docker stack*, this problem could be eliminated.

The employed DNS solution can be further improved by implementing a second DNS for caching purposes. This caching DNS would be employed in Porto to improve domain-name resolution times for the network managers.

The developed system allows for the integration of many more services, for example, an e-mail server. In the future, an increased focus on this area would likely prove fruitful. Since all traffic flow (in and out) goes through the firewall, it is possible to implement an Intrusion-Detection System (for example, *Snort* [2]) and check for malicious connections.

X. Conclusion

The group was able to build a system that allows for a configurable, malleable, and extensible network. The pipeline automates the deployment of all components (and their configurations), and when paired with the version control system adopted, allows us to easily switch between version of the code/system.

The deployed web application is accessible by clients inside and outside the company's network, and can be fully monitored off-site. The network is horizontally scalable, through the use of load-balancers for the web application. Furthermore, nagios can be used to assess whether or not the network can handle the incoming traffic, as well as identifying critical components that may hinder the network's performance.

[2]Snort Intrusion-Detection System main page

REFERENCES

[1] Linux Server project's WireGuard container (and associated documentation) on DockerHub. Accessed 9 jun, 2022.

[2] Jason Rivers, 3rd party Nagios container, on DockerHub. Accessed 9 jun, 2022.

[3] Arch Linux wiki's article on ISC DHCPCD. Accessed 9 jun, 2022.

[4] Arch Linux wiki's article on ISC DHCPD. Accessed 9 jun, 2022.

XI. Appendix

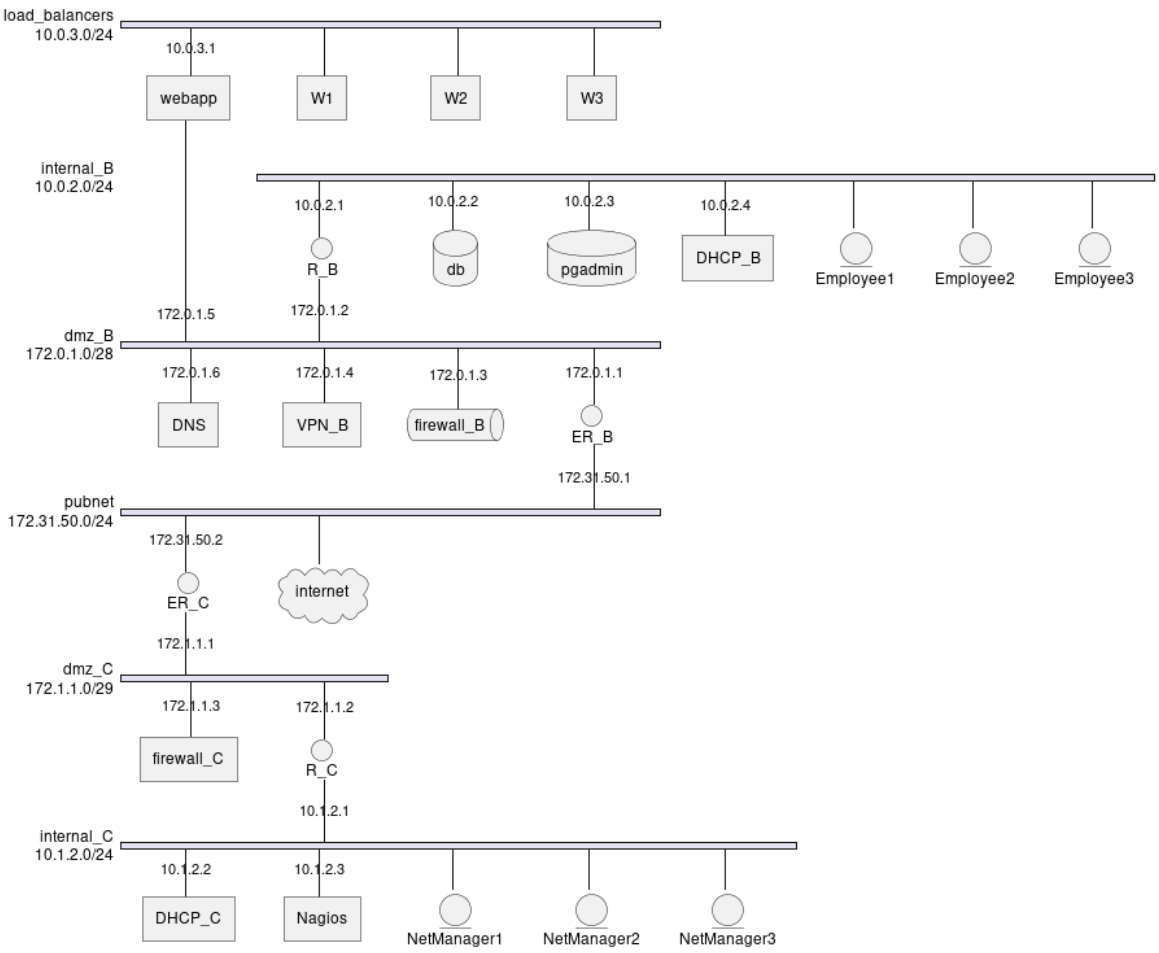

Fig. 1: Network Diagram

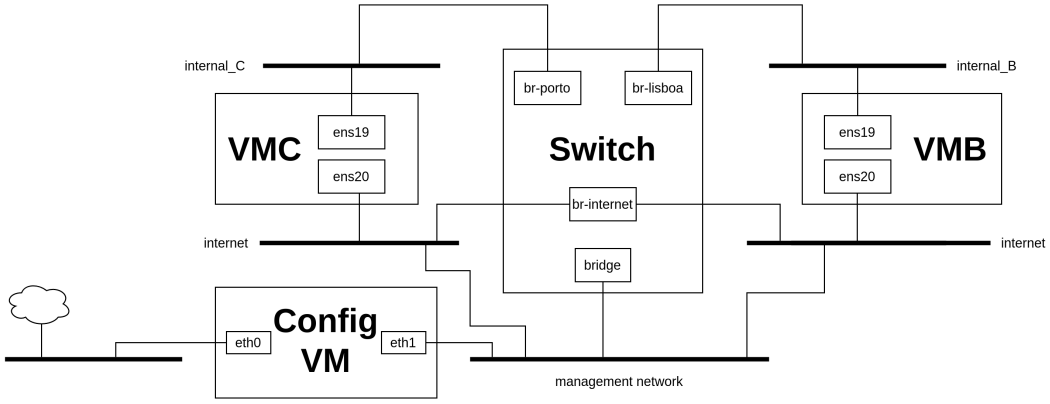

Fig. 2: Environment Diagram