Virtualization and cloud computing

- Enrico Pezzano 4825087
- Camilla Magistrello 4512554
- Filippo Gavuglio 4875323

Project goal

- Running a set of services and monitoring them
- Run easly on different system
- Scalable and multi-node

```
[all]
target1 ansible_host=127.0.0.1
    ansible_port=2201 ansible_user=vagrant
    ansible_ssh_private_key_file=/home/enriicola/Desktop/exam-vccamillino/.vagrant/machines/target1/vmware_workstation/private_key
target2 ansible_host=127.0.0.1
    ansible_port=2201 ansible_user=vagrant
    ansible_ssh_private_key_file=/home/enriicola/Desktop/exam-vccamillino/.vagrant/machines/target2/vmware_workstation/private_key
```

The Inventory problem:

One of the first problem we encounter was understanding the role of the inventory file and how to use it properly

A first (and dumb) approach was to hard-code the ansible configuration directly to the inventory file

```
[targets]
target[1:2].vcc.local

[targets:vars]
ansible_user=vagrant
```

The inventory solution

The simple solution was to use direcly the ssh vagrant user to tell ansible the node to connect

The blob docker build problem

In the «Build and push container images» task, the task couldn’t finish due to a blob error

```yaml
- name: Build and push container images
  community.docker.docker_image:
    name: "registry.vcc.internal:5000/{{ item }}"
    tag: latest
    source: build
    build:
      path: "/opt/container-images/{{ item }}"
    push: true
    state: present
  loop: "{{ swarm_build_images }}"
  # retry the task that may fail
  retries: 5
  delay: 3
  register: result
  until: result is not failed
```

```yaml
- name: Build container images
  community.docker.docker_image:
    name: "registry.vcc.internal:5000/{{ item }}"
    tag: latest
    source: build
    build:
      path: "/opt/container-images/{{ item }}"
    state: present
  loop: "{{ swarm_build_images }}"
  retries: 3
  delay: 3

- name: Wait for Docker to sync layers
  ansible.builtin.pause:
    seconds: 3

- name: Push container images to registry
  community.docker.docker_image:
    name: "registry.vcc.internal:5000/{{ item }}"
    tag: latest
    source: local
    push: true
    state: present
  loop: "{{ swarm_build_images }}"
  retries: 5
  delay: 3
  register: result
  until: result is not failed
```

The blob docker build solution

The solution was to split the task in 3 sub-task, to give docker the time to build the images completely

```
# Ensure network interface is configured
node.vm.provision :shell, :inline => <<-SHELL
  if ! ip a show enp0s8 | grep -q '#{MYNET}.1#{i}/24'; then
    ip addr add #{MYNET}.1#{i}/24 dev enp0s8 || true
  fi
SHELL
```

The network interface problem

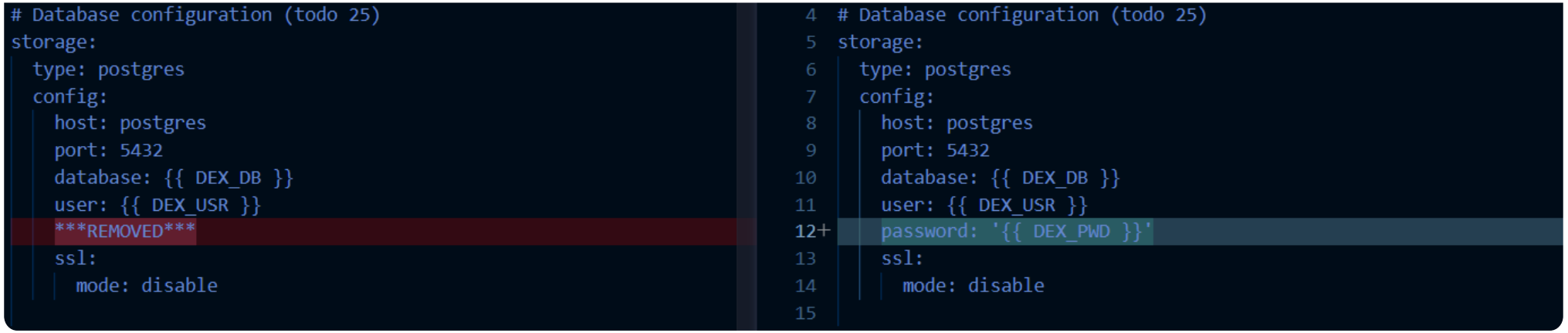
During the provisioning phase, the network interface name ‘enp0s8’ only worked on Linux.

```
# Ensure network interface is configured (auto-detect interface name)
node.vm.provision :shell, :inline => <<-SHELL
  IFACE=$(ip -o link show | awk -F': ' '$2 ~ /^(enp0s8|eth1)$/ {print $2; exit}')
  if [ -n "$IFACE" ] && ! ip a show $IFACE | grep -q '#{MYNET}.1#{i}/24'; then
    ip addr add #{MYNET}.1#{i}/24 dev $IFACE || true
  fi
SHELL
```

The network interface solution

Obviously, on a PC with network interfaces using different identifiers, the enp0s8 interface is not found and the provisioning process fails.

We replaced it with a command that lists the available network interface names and uses one of the existing ones.



```
# Database configuration (todo 25)
storage:
  type: postgres
  config:
    host: postgres
    port: 5432
    database: {{ DEX_DB }}
    user: {{ DEX_USR }}
    ***REMOVED***
    ssl:
      mode: disable
```

```
4  # Database configuration (todo 25)
5  storage:
6    type: postgres
7    config:
8      host: postgres
9      port: 5432
10     database: {{ DEX_DB }}
11     user: {{ DEX_USR }}
12+    password: '{{ DEX_PWD }}'
13     ssl:
14       mode: disable
15
```

The security problem

Since several passwords at some point in development were written in the git history, we needed to clean it from any hard-coded passwords

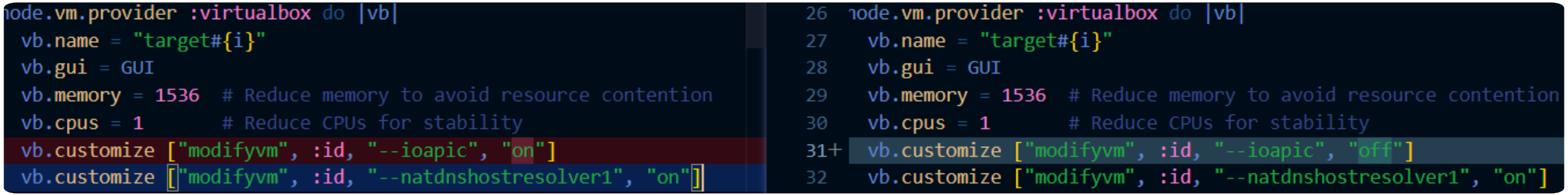
```
docker_registry_password: "{{ lookup('ansible.builtin.password', 'credentials/container-registry-password', length=32) }}"
postgres_password: "{{ lookup('ansible.builtin.password', 'credentials/postgres-password', length=32) }}"

# Main postgres credentials
DB_USR: postgres
DB_PWD: "{{ lookup('ansible.builtin.password', 'credentials/postgres-password', length=32) }}"

# Dex database credentials (todo 14a)
DEX_USR: dex_user
DEX_PWD: "{{ lookup('ansible.builtin.password', 'credentials/dex-db-password', length=32) }}"
DEX_DB: dex_db

# Forgejo database credentials (todo 14b)
FOR_USR: forgejo_user
FOR_PWD: "{{ lookup('ansible.builtin.password', 'credentials/forgejo-db-password', length=32) }}"
FOR_DB: forgejo_db
FOR_SCR: "{{ lookup('ansible.builtin.password', 'credentials/forgejo-secret', length=64) }}"
FOR_EMAIL: admin@vcc.internal

# Forgejo configuration secrets (todo 31)
FOR_JWT_SCR: "{{ lookup('ansible.builtin.password', 'credentials/forgejo-jwt-secret', length=64) }}"
FOR_SCR_KEY: "{{ lookup('ansible.builtin.password', 'credentials/forgejo-secret-key', length=64) }}"
FOR_CK_USR: vcc_git_username
FOR_RMB_NM: vcc_git_remember_me
FOR_INT_TOK: "{{ lookup('ansible.builtin.password', 'credentials/forgejo-internal-token', length=64) }}"
```

The secret.yml file

Once the previously hard-coded passwords were removed, we added a secrets.yml file which, through Ansible Vault, allowed us to avoid storing them in plain text.

The kernel panic problem

For most of the time we worked on a Linux host, but when we moved to a Windows host, at a certain point during the execution of the Vagrantfile we encountered a kernel panic.



The kernel panic solution

The solution was to disable the VM's I/O APIC, since on Windows it caused conflicts with Hyper-V.

The choose of VirtualBox

Since the VMware plugin on the Linux machine caused issues, we opted for VirtualBox, as it does not require additional plugins for Vagrant.

Thanks