

# Задание от Группы разработки безопасных сетевых решений

## Задание:

Необходимо развернуть микросервис на виртуальной машине (ВМ), используя систему управления конфигурациями и описать выполненные действия.

- В качестве гипервизора 2-го типа разрешается использовать любой, по своему усмотрению: VMware Workstation, VirtualBox, HyperV, Qemu, Vagrant и т.п.  
Операционная система для ВМ: RedHat-подобные системы, например, Rocky Linux, AlmaLinux и т.д.
- Микросервис должен доставляться на виртуальную машину с помощью системы управления конфигурациями, после чего микросервис должен работать в фоне на ВМ. Необходимая CMS (Configuration Management Systems / Система Управления Конфигурациями): Ansible.
- Микросервис: HTTP сервер, который экспортирует на 8080 порт Prometheus метрики. Одна из метрик должна предоставлять данные о том, на какой типе хоста запущен сервер: виртуальная машина, контейнер или физический сервер. Для написания можно использовать любой язык программирования (желательно, Python или Golang). Рекомендуется использовать готовые библиотеки для поднятия HTTP сервера и проброса Prometheus метрик.
- В браузере на виртуальной машине по адресу `http://localhost:8080` должны отображаться соответствующие Prometheus метрики.

## Решение

1. В качестве гипервизора я использовал VirtualBox. Операционная система: Ubuntu 22.04.1 LTS.
2. Написал микросервис на python. Использовал фреймворк Flask.

```
from flask import Flask # Импортируем класс Flask для создания веб-приложения
from prometheus_client import Counter, generate_latest, CONTENT_TYPE_LATEST #
Импортируем Counter для подсчета метрик и функции для генерации Prometheus
ответа
import os # Импортируем модуль os для работы с файловой системой

app = Flask(__name__) # Создаем экземпляр Flask-приложения
```

```

requests_total = Counter('requests_total', 'Total HTTP Requests') # Создаем
счетчик "requests_total" для подсчета общих HTTP-запросов

def get_host_type(): # Функция для определения типа хоста
    if os.path.exists('/.dockerenv'): # Проверяем, существует ли файл
        ".dockerenv", который указывает на контейнер Docker
        return 'container'
    elif os.path.exists('/proc/vz'): # Проверяем, существует ли каталог
        "/proc/vz", который указывает на виртуальную машину
        return 'virtual_machine'
    else: # Если ни один из файлов не найден, предполагаем, что это
        физический сервер
        return 'physical_server'

@app.route('/')
def metrics():
    requests_total.inc() # Увеличиваем счетчик при каждом запросе
    response = generate_latest(requests_total) # Генерируем ответ Prometheus
    response += f'host_type {get_host_type()}\n'.encode() # Добавляем к
    ответу информацию о типе хоста
    return response, 200, {'Content-Type': CONTENT_TYPE_LATEST} # Возвращаем
    ответ с кодом 200 (OK) и заголовком Content-Type

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080) # Запускаем приложение на всех
    доступных интерфейсах (0.0.0.0) на порту 8080

```

### 3. Установил venv пакет

```
sudo apt install python3-venv
```

Создал venv

```
python3 -m venv my_venv
```

Активировал venv

```
source my_venv/bin/activate
```

Установил зависимости

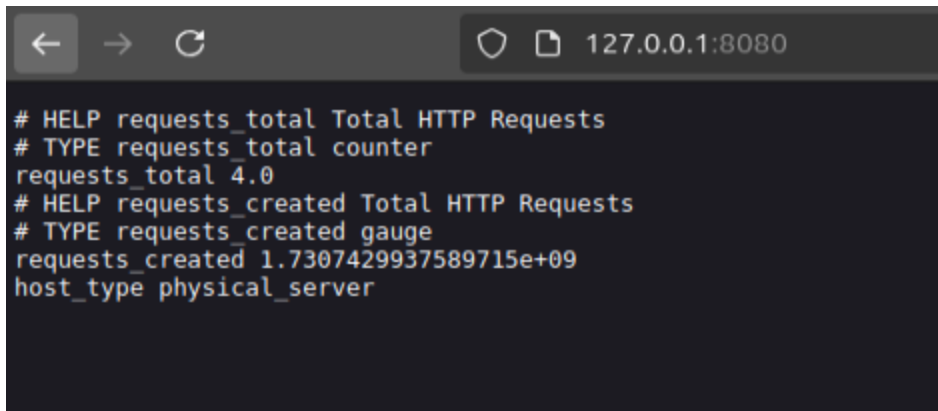
```
pip install flask
```

```
pip install prometheus_client
```

Запустил микросервис

```
python main.py
```

Убедился, что микросервис работает

A screenshot of a web browser window displaying Prometheus metrics. The address bar shows '127.0.0.1:8080'. The page content lists several metrics: 'requests\_total' (Total HTTP Requests, counter, value 4.0), 'requests\_created' (Total HTTP Requests, gauge, value 1.7307429937589715e+09), and 'host\_type' (physical\_server).

```
# HELP requests_total Total HTTP Requests  
# TYPE requests_total counter  
requests_total 4.0  
# HELP requests_created Total HTTP Requests  
# TYPE requests_created gauge  
requests_created 1.7307429937589715e+09  
host_type physical_server
```

#### 4. Создал стандартный ansible.cfg

```
[defaults]  
remote_user = vboxuser  
host_key_checking = False  
stdout_callback = yaml  
  
[privilege_escalation]  
become = True  
become_method = sudo  
become_user = root  
become_ask_pass = False
```

Создал hosts

```
localhost ansible_connection=local
```

Я буду разворачивать микросервис на localhost. Для деплоя на другие сервера, нужно отредактировать соответствующим образом hosts и по необходимости отредактировать ansible.cfg.

Создал ansible плейбук playbook.yml

```
- hosts: all  
become: yes  
tasks:  
  - name: Install Python dependencies # устанавливаем python3 и pip через apt
```

```
apt:
name:
- python3
- python3-pip
state: present

- name: Copy application script to VM
copy: # копируем файл с программой в /opt/micro/main.py
src: ./main.py
dest: /opt/micro/main.py
mode: '0777'

- name: venv
command: python3 -m venv /opt/micro/venv # создаем venv
- name: Install Python packages in virtual environment
pip: # через pip устанавливаю flask и prometheus_client в venv
name:
- flask
- prometheus_client
virtualenv: "/opt/micro/venv"
state: present

- name: Create systemd service file for microservice # создаю юнит для автозапуска микросервиса
copy:
dest: /etc/systemd/system/micro.service
content: |
[Unit]
Description=A simple HTTP server for Prometheus metrics

[Service]
ExecStart=/opt/micro/venv/bin/python3 /opt/micro/main.py # запускаю main.py через venv
Restart=always

[Install]
WantedBy=multi-user.target

- name: Start and enable microservice service
systemd:
name: micro
```

```
state: started # запускаю юнит
enabled: yes # добавляю юнит в автозапуск
```

## 5. Запускаю playbook

```
sudo ansible-playbook -i hosts playbook.yml
```

Проверяю юнит

```
sudo systemctl status micro
```

```
● micro.service - A simple HTTP server for Prometheus metrics
   Loaded: loaded (/etc/systemd/system/micro.service; enabled; preset: enabled)
   Active: active (running) since Mon 2024-11-04 17:56:33 UTC; 1h 58min ago
     Main PID: 77364 (python3)
        Tasks: 1 (limit: 10126)
      Memory: 20.2M (peak: 20.8M)
         CPU: 1.241s
    CGroup: /system.slice/micro.service
            └─77364 /opt/micro/venv/bin/python3 /opt/micro/main.py

Nov 04 17:56:33 ubuntu7 python3[77364]: * Debug mode: off
Nov 04 17:56:33 ubuntu7 python3[77364]: WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
Nov 04 17:56:33 ubuntu7 python3[77364]: * Running on all addresses (0.0.0.0)
Nov 04 17:56:33 ubuntu7 python3[77364]: * Running on http://127.0.0.1:8080
Nov 04 17:56:33 ubuntu7 python3[77364]: * Running on http://10.0.2.15:8080
Nov 04 17:56:33 ubuntu7 python3[77364]: Press CTRL+C to quit
Nov 04 17:56:35 ubuntu7 python3[77364]: 127.0.0.1 - - [04/Nov/2024 17:56:35] "GET / HTTP/1.1" 200 -
Nov 04 17:56:36 ubuntu7 python3[77364]: 127.0.0.1 - - [04/Nov/2024 17:56:36] "GET / HTTP/1.1" 200 -
Nov 04 17:56:36 ubuntu7 python3[77364]: 127.0.0.1 - - [04/Nov/2024 17:56:36] "GET / HTTP/1.1" 200 -
Nov 04 17:56:36 ubuntu7 python3[77364]: 127.0.0.1 - - [04/Nov/2024 17:56:36] "GET / HTTP/1.1" 200 -
(venv) vboxuser@ubuntu7:~/micro$
```

Проверяю <http://127.0.0.1> в браузере

```
← → ↻ 127.0.0.1:8080

# HELP requests_total Total HTTP Requests
# TYPE requests_total counter
requests_total 11.0
# HELP requests_created Total HTTP Requests
# TYPE requests_created gauge
requests_created 1.7307429937589715e+09
host_type physical_server
```

После обновления счетчик запросов увеличивается

```
← → ↻ 127.0.0.1:8080

# HELP requests_total Total HTTP Requests
# TYPE requests_total counter
requests_total 12.0
# HELP requests_created Total HTTP Requests
# TYPE requests_created gauge
requests_created 1.7307429937589715e+09
host_type physical_server
```

## Бонусное задание 1:

- Адаптировать ansible-role и ansible-playbook также под альтернативный сценарий: запуск того же микросервиса в контейнере на той же ВМ.
- В качестве CRI (Container Runtime Interface) использовать Docker или Podman.

- Установку докера в ansible-role закидывать необязательно, можно выполнить руками заранее.

## Решение

1. Я написал requirements.txt для main.py

```
flask
prometheus_client
```

2. Создал Dockerfile

```
FROM python:3.8-slim-buster

\# Устанавливаем зависимости
COPY requirements.txt /requirements.txt
RUN pip install -r /requirements.txt

\# Копируем файлы приложения
COPY ./main.py /app/main.py

\# Запускаем приложение
CMD ["python", "/app/main.py"]
```

3. Проверка Dockerfile. Собрал образ.

```
docker build -t micro .
```

Запустил контейнер

```
docker run -d micro
```

Проверил через браузер, что приложение запущено

4. Создал ansible роль, которая будет собирать образ и запускать контейнер.

```
mkdir -p ./roles/docker-micro-service/tasks
nano ./roles/docker-micro-service/tasks/main.yml
```

```
- `name: Build Docker image` # сборка образа
  `docker_image:`
    `name: micro-service:latest`
    `source: build`
    `build:`
      `path: ./` # путь до Dockerfile относительно директории запуска плейбука
- `name: Run Docker container` # запуск контейнера
```

```
`docker_container:`  
  `name: micro-service`  
  `image: micro-service:latest`  
  `state: started`  
  `ports:`  
    - `"8080:8080"` # проброс 8080 порта контейнера на 8080 порт хоста  
  `restart_policy: always`
```

## 5. Создал ansible playbook

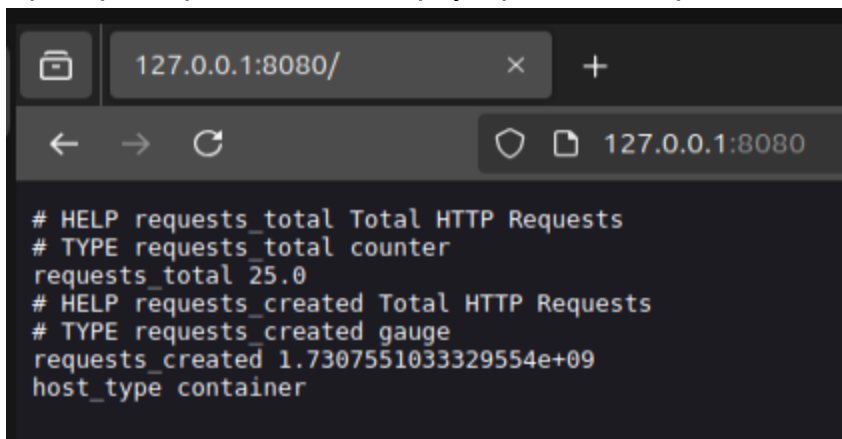
```
nano container_playbook.yml
```

```
- hosts: all  
  become: yes  
  roles:  
    - docker-micro-service # вызов роли
```

## 6. Запускаю плейбук

```
sudo ansible-playbook -i hosts container_playbook.yml
```

Проверяю приложение из браузера, счетчик работает.



Директория проекта:

```
vboxuser@ubuntu7:~/micro$ ls -la  
total 44  
drwxrwxr-x  4 vboxuser vboxuser 4096 Nov  4 21:44 .  
drwxr-x--- 33 vboxuser vboxuser 4096 Nov  4 19:10 ..  
-rw-rw-r--  1 vboxuser vboxuser  185 Nov  4 16:44 ansible.cfg  
-rw-rw-r--  1 vboxuser vboxuser   63 Nov  4 21:03 container_playbook.yml  
-rw-rw-r--  1 vboxuser vboxuser  312 Nov  4 20:27 Dockerfile  
-rw-rw-r--  1 vboxuser vboxuser   35 Nov  4 16:58 hosts  
-rw-rw-r--  1 vboxuser vboxuser  694 Nov  4 16:27 main.py  
drwxrwxr-x  5 vboxuser vboxuser 4096 Nov  4 16:04 my_venv  
-rw-rw-r--  1 vboxuser vboxuser 1078 Nov  4 17:55 playbook.yml  
-rw-rw-r--  1 vboxuser vboxuser   24 Nov  4 20:26 requirements.txt  
drwxrwxr-x  3 vboxuser vboxuser 4096 Nov  4 20:38 roles
```

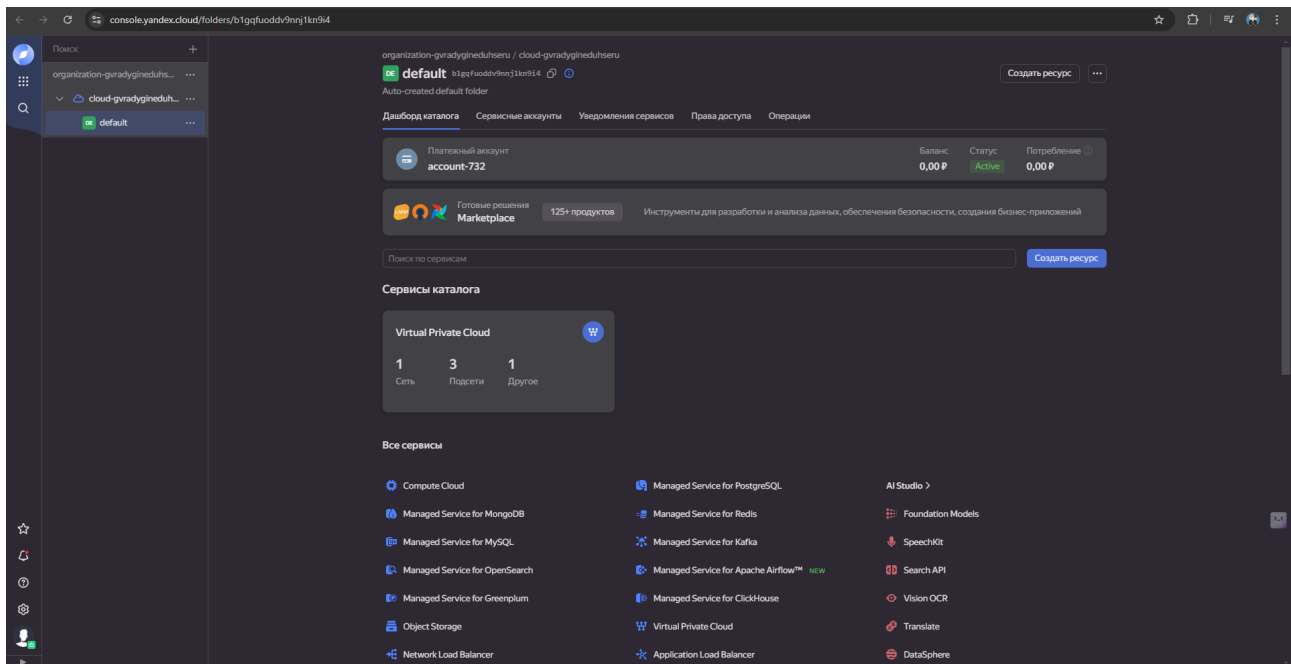
## Бонусное задание 2:

- Автоматизировать этап создания виртуальной машины из пункта 1 основного задания. Автоматизация должна включать в себя (1) создание виртуальной машины (2) установку ОС в виртуальной машине (3) настройку ОС в виртуальной машине (до этапа разворачивания микросервиса, но, по возможности, с автоматизацией установки и настройки CRI).
- В качестве основы для автоматизации создания VM можно использовать любые языки программирования либо утилиты для реализации подхода Infrastructure as a Code (во втором случае предпочтительно Terraform). Если VM создается на платформе ОС Windows, то автоматизацию создания VM предпочтительно реализовать с использованием Powershell.

## Решение

Для решения данной задачи я буду использовать [Terraform + Yandex.Cloud](#)

### 1. У меня уже есть аккаунт Yandex.Cloud



### 2. Скачиваю - [terraform\\_1.9.8\\_linux\\_amd64.zip](#)

Добавляю путь до terraform в PATH

```
export PATH=$PATH:/home/vboxuser/Downloads/terraform_1.9.8_linux_amd64
```

Открою ~/.bashrc

```
nano ~/.bashrc
```

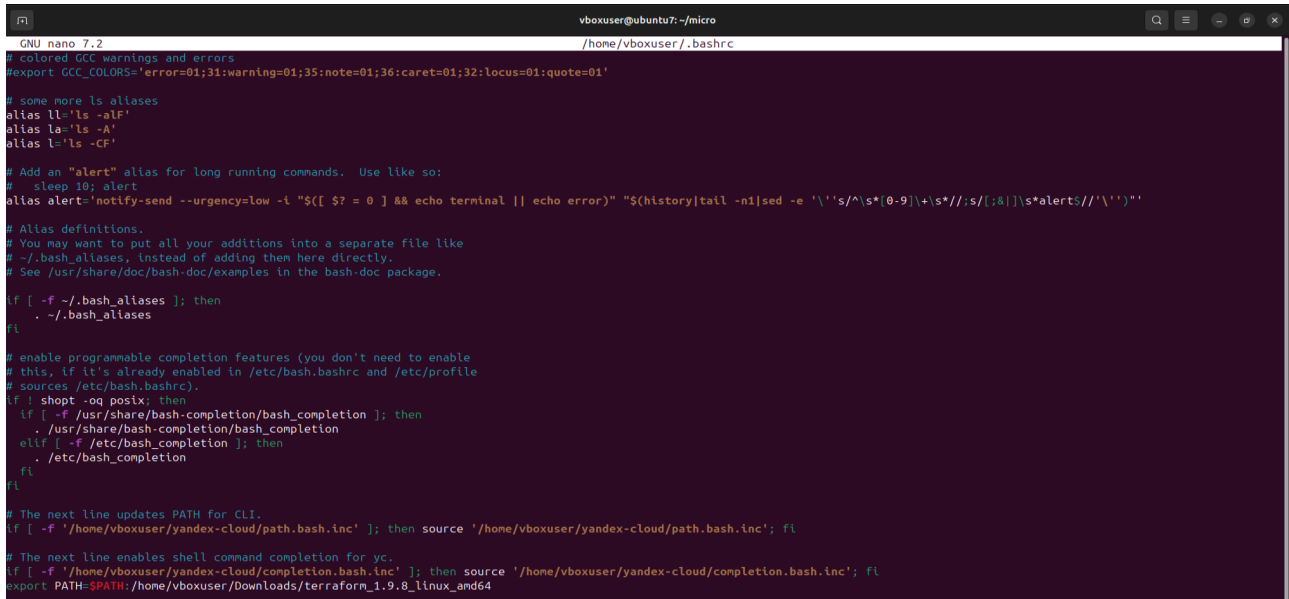
Добавлю export

```
PATH=$PATH:/home/vboxuser/Downloads/terraform_1.9.8_linux_amd64
```

 в конце



файла, чтобы путь до terraform всегда добавлялся к PATH после перезагрузки виртуальной машины.



```
GNU nano 7.2 /home/vboxuser/.bashrc
# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -lF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands. Use like so:
# sleep 10; alert
alias alert='notify-send --urgency=low -i "${?} = 0" && echo terminal || echo error)' "${history|tail -n1|sed -e '\s/^s*[0-9]\|+\s*///;s/[:;&]]\s*alert$//'\`'"

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

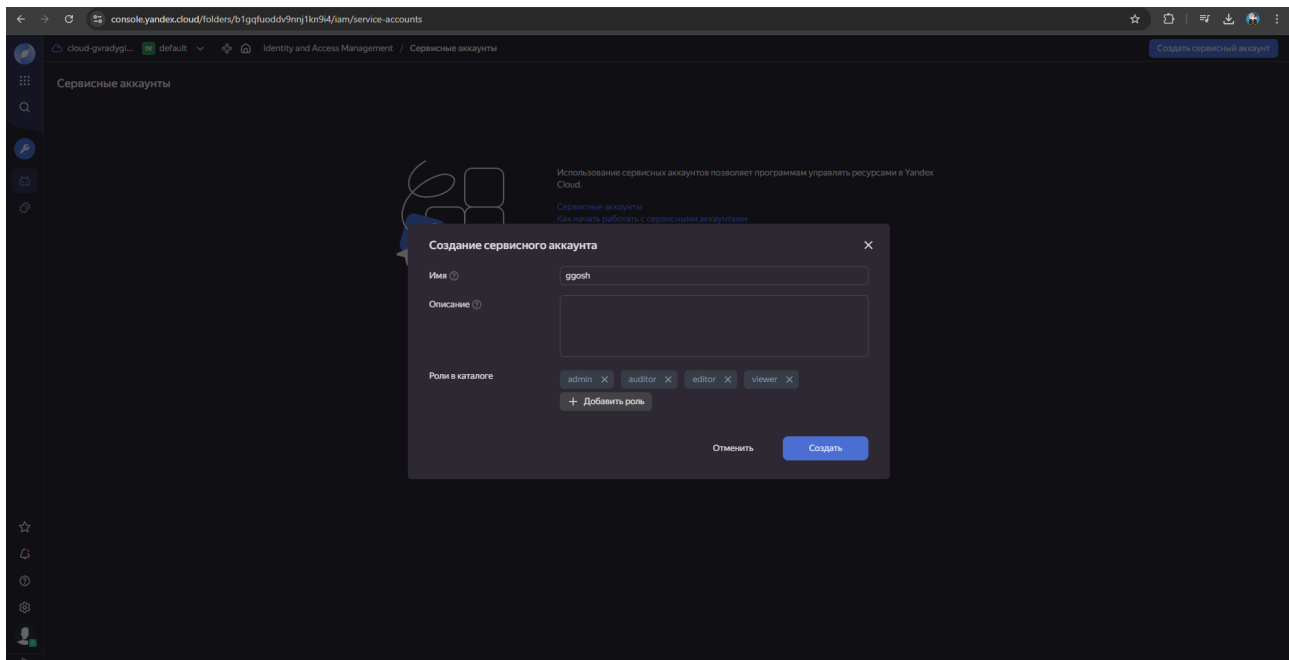
# The next line updates PATH for CLI.
if [ -f '/home/vboxuser/yandex-cloud/path.bash.inc' ]; then source '/home/vboxuser/yandex-cloud/path.bash.inc'; fi

# The next line enables shell command completion for yc.
if [ -f '/home/vboxuser/yandex-cloud/completion.bash.inc' ]; then source '/home/vboxuser/yandex-cloud/completion.bash.inc'; fi
export PATH=$PATH:/home/vboxuser/Downloads/terraform_1.9.8_linux_and64
```

Обновлю конфигурацию bash

```
source ~/.bashrc
```

3. Создал сервисный аккаунт в консоли Yandex.Cloud с ролями admin, auditor, editor, viewer.



4. Скачал Интерфейс командной строки Yandex Cloud (CLI)

```
curl -sSL https://storage.yandexcloud.net/yandexcloud-yc/install.sh | bash
```

Получил [Oauth token](#) в сервисе Яндекс ID

Выполняю аутентификацию

```
yc init
```

Вставляю Oauth токен, выбираю настройки по дефолту.

Создал авторизованный ключ для сервисного аккаунта и запишите его файл

```
yc iam key create \ --service-account-id <идентификатор_сервисного_аккаунта> \ --folder-name default \ --output key.json
```

Задал конфигурацию профиля

```
yc config set service-account-key key.json
yc config set cloud-id b1geel1ciiqmi98hs44s
yc config set folder-id b1gqfuoddv9nnj1kn9i4
```

Добавил аутентификационные данные в переменные окружения

```
export YC_TOKEN=$(yc iam create-token) export YC_CLOUD_ID=
(yc config get cloud-id) \ export YC_FOLDER_ID=(yc config get folder-id)
```

Открыл файл конфигурации Terraform CLI

```
nano ~/.terraformrc
```

Добавил в него

```
provider_installation {
  network_mirror {
    url = "https://terraform-mirror.yandexcloud.net/"
    include = ["registry.terraform.io/*/*"]
  }
  direct {
    exclude = ["registry.terraform.io/*/*"]
  }
}
```

5. Создал в директории проекта конфигурационный файл provider.tf

```
nano provider.tf
```

Добавил в него

```
terraform {
  required_providers {
    yandex = {
      source = "yandex-cloud/yandex" # указываем провайдера
    }
  }
  required_version = ">= 0.13" # версия провайдера
}

provider "yandex" {
  zone = "ru-central1-a" # зона доступности по умолчанию
  folder_id = "b1gqfuoddv9nnj1kn9i4"
}
```

6. Создал main.tf, в котором описал создание сети, подсети, виртуальной машины

```
        # Создание VPC и подсети
resource "yandex_vpc_network" "this" { #сеть
    name = "private"
}

resource "yandex_vpc_subnet" "private" { #подсеть
    name            = "private"
    zone            = "ru-central1-a"
    v4_cidr_blocks = ["192.168.10.0/24"]
    network_id      = yandex_vpc_network.this.id
}

resource "yandex_vpc_address" "addr" { # Создание внешнего IP-адреса
    name = "vm-adress"
    external_ipv4_address {
        zone_id = "ru-central1-a"
    }
}

# Создание диска и виртуальной машины
resource "yandex_compute_disk" "boot_disk" {
    name      = "boot-disk"
    zone      = "ru-central1-a"
    image_id  = "fd8ba9d5mfvlncnt2kd" # Ubuntu 22.04 LTS
    size      = 10
}

# Создает виртуальную машину с именем "linux-vm"
resource "yandex_compute_instance" "this" {
    name                        = "linux-vm"
    allow_stopping_for_update = true
    platform_id                = "standard-v3"
    zone                        = "ru-central1-a"

    resources { #2 ядра и 4 ГБ памяти
        cores   = "2"
        memory  = "4"
    }
}
```

```

boot_disk { #загрузочный диск для виртуальной машины
  disk_id = yandex_compute_disk.boot_disk.id
}

network_interface { # Подключаем виртуальную машину к подсети и настраиваем
NAT
  subnet_id = yandex_vpc_subnet.private.id
  nat      = true
  nat_ip_address = yandex_vpc_address.addr.external_ipv4_address[0].address
}

# Задаёт метаданные для виртуальной машины, которые берутся из файла
meta.yml
metadata = {
  user-data = "${file("./meta.yml")}"
}
}

```

7. Создал meta.yml. Инструкции из этого файла будут выполняться при создании виртуальной машины.

```

#cloud-config
datasource:
  Ec2:
    strict_id: false
ssh_pwauth: no
users:
- name: admin # создаем пользователя admin
  sudo: ALL=(ALL) NOPASSWD:ALL # Предоставить полные права sudo без пароля
  shell: /bin/bash
  ssh_authorized_keys: # указываем публичный ключ, который сохраняется на
сервере
- ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIDU+qfeQcgyub4xTRjs1M9Xe4XPkBLuiV1JPZGq7T89I
vboxuser@ubuntu7
write_files: # установка docker
- path: "/usr/local/etc/docker-start.sh"
  permissions: "755"
  content: |

```

```
#!/bin/bash

# Docker
echo "Installing Docker"
sudo apt update -y && sudo apt install docker.io -y
echo "Grant user access to Docker"
sudo usermod -aG docker ${USER}
newgrp docker
defer: true

runcmd:
  - [su, admin, -c, "/usr/local/etc/docker-start.sh"] # запуск установки
docker
```

8. На сервере нет Dockerfile и приложения, поэтому я решил изменить ansible role, добавив туда копирование main.py, requirements.txt, Dockerfile.

```
nano roles/docker-micro-service/tasks/main.yml
```

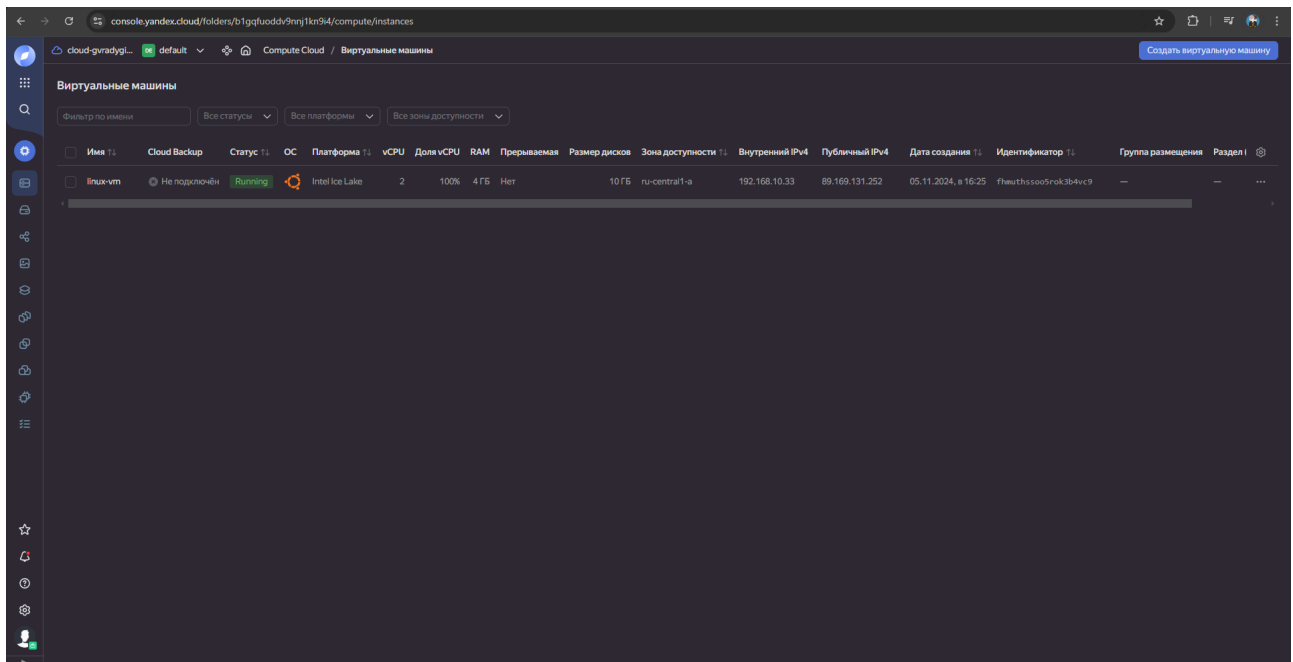
```
- name: copy main.py # копируем main.py
  copy:
    src: main.py
    dest: ./main.py
- name: copy requirements.txt # копируем requirements.txt
  copy:
    src: requirements.txt
    dest: ./requirements.txt
- name: copy Dockerfile # копируем Dockerfile
  copy:
    src: Dockerfile
    dest: ./Dockerfile
- name: Build Docker image # собираем образ
  docker_image:
    name: micro-service:latest
    source: build
    build:
      path: ./
- name: Run Docker container # запускаем контейнер на 8080 порту
  docker_container:
    name: micro-service
    image: micro-service:latest
    state: started
```

```
ports:
  - "8080:8080"
restart_policy: always
```

## 9. Создадим виртуальную машину в облаке:

```
terraform apply
```

Скопируем из консоли Yandex.Cloud публичный IP



Отредактируем файл hosts

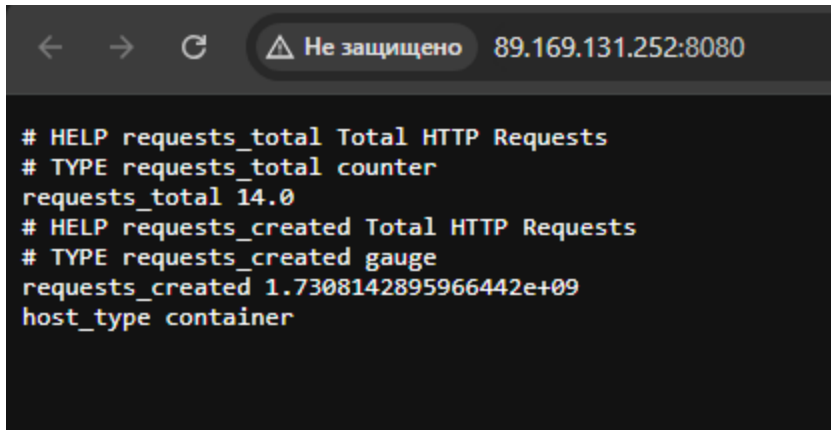
```
nano hosts
```

```
[all]
89.169.131.252 ansible_ssh_private_key_file=/home/vboxuser/.ssh/id_ed25519
ansible_user=admin
```

## 10. Запустим playbook

```
sudo ansible-playbook -i hosts container_playbook.yml
```

Проверим что приложение запустилось и публично доступно.



```
# HELP requests_total Total HTTP Requests
# TYPE requests_total counter
requests_total 14.0
# HELP requests_created Total HTTP Requests
# TYPE requests_created gauge
requests_created 1.7308142895966442e+09
host_type container
```

11. Но зачем вручную смотреть публичный IP и добавлять его в hosts, если это можно автоматизировать. Отредактируем main.tf

```
nano main.tf
```

```
# Создание VPC и подсети
resource "yandex_vpc_network" "this" {
  name = "private"
}

resource "yandex_vpc_subnet" "private" {
  name           = "private"
  zone           = "ru-central1-a"
  v4_cidr_blocks = ["192.168.10.0/24"]
  network_id     = yandex_vpc_network.this.id
}

resource "yandex_vpc_address" "addr" {
  name = "vm-address"
  external_ipv4_address {
    zone_id = "ru-central1-a"
  }
}

# Создание диска и виртуальной машины
resource "yandex_compute_disk" "boot_disk" {
  name     = "boot-disk"
  zone     = "ru-central1-a"
  image_id = "fd8ba9d5mfvlncknt2kd" # Ubuntu 22.04 LTS
}
```

```

    size      = 10
}

resource "yandex_compute_instance" "this" {
  name                  = "linux-vm"
  allow_stopping_for_update = true
  platform_id          = "standard-v3"
  zone                 = "ru-central1-a"

  resources {
    cores   = "2"
    memory  = "4"
  }

  boot_disk {
    disk_id = yandex_compute_disk.boot_disk.id
  }

  network_interface {
    subnet_id = yandex_vpc_subnet.private.id
    nat       = true
    nat_ip_address = yandex_vpc_address.addr.external_ipv4_address[0].address
  }

  # перезаписываем файл hosts2 для ansible в нужном формате
  provisioner "local-exec" {
    command = "echo \"${self.network_interface.0.nat_ip_address}\"
ansible_ssh_private_key_file=/home/vboxuser/.ssh/id_ed25519 ansible_user=admin
\n \" > hosts2"
  }

  metadata = {
    user-data = "${file("./meta.yml")}"
  }
}

output "public_ip" {
  value = yandex_compute_instance.this.network_interface.0.nat_ip_address
}

```



## 12. Пересоздадим инфраструктуру и запустим плейбук

```
terraform destroy
```

```
terraform apply
```

```
sudo ansible-playbook -i hosts2 container_playbook.yml
```

Иногда могут получаться такие ошибки:

```
vboxuser@ubuntu7:~/micro$ sudo ansible-playbook -i hosts2 container_playbook.yml

PLAY [all] *****

TASK [Gathering Facts] *****
fatal: [84.201.134.31]: UNREACHABLE! => changed=false
  msg: 'Failed to connect to the host via ssh: ssh: connect to host 84.201.134.31 port 22: Connection timed out'
  unreachable: true

PLAY RECAP *****
84.201.134.31      : ok=0    changed=0    unreachable=1    failed=0    skipped=0    rescued=0    ignored=0

vboxuser@ubuntu7:~/micro$ sudo ansible-playbook -i hosts2 container_playbook.yml

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [84.201.134.31]

TASK [docker-micro-service : copy main.py] *****
ok: [84.201.134.31]

TASK [docker-micro-service : copy requirements.txt] *****
ok: [84.201.134.31]

TASK [docker-micro-service : copy Dockerfile] *****
ok: [84.201.134.31]

TASK [docker-micro-service : Build Docker image] *****
fatal: [84.201.134.31]: FAILED! => changed=false
  msg: 'Error connecting: Error while fetching server API version: ("Connection aborted.", FileNotFoundError(2, "No such file or directory"))'

PLAY RECAP *****
84.201.134.31      : ok=4    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
```

В таком случае нужно повторно выполнять запуск, через 1-3 перезапуска ошибки уйдут. Понять причину их возникновения мне не удалось.

```
sudo ansible-playbook -i hosts2 container_playbook.yml
```

13. Такой подход к деплою может быть неудобен, потому что при изменении структуры проекта нужно менять ansible role, добавляя копирование новых файлов. Кроме того, нет никакого хранилища, где могли бы располагаться старые сборки. Поэтому лучше создавать в main.tf Yandex Container Registry, собирать образ вручную у себя на ПК, через ansible playbook делать pull из Container Registry на сервере и запускать контейнер. Так мы обеспечим большую изолированность и надежность.

Добавил блок с созданием registry в main.tf

```
nano main.tf
```

```
# Создание VPC и подсети
resource "yandex_vpc_network" "this" {
  name = "private"
}

resource "yandex_vpc_subnet" "private" {
  name           = "private"
  zone           = "ru-central1-a"
  v4_cidr_blocks = ["192.168.10.0/24"]
}
```

```

    network_id      = yandex_vpc_network.this.id
}

resource "yandex_vpc_address" "addr" {
  name = "vm-address"
  external_ipv4_address {
    zone_id = "ru-central1-a"
  }
}

# Создание диска и виртуальной машины
resource "yandex_compute_disk" "boot_disk" {
  name      = "boot-disk"
  zone      = "ru-central1-a"
  image_id  = "fd8ba9d5mfvlncnt2kd" # Ubuntu 22.04 LTS
  size      = 10
}

resource "yandex_compute_instance" "this" {
  name              = "linux-vm"
  allow_stopping_for_update = true
  platform_id       = "standard-v3"
  zone              = "ru-central1-a"

  resources {
    cores   = "2"
    memory  = "4"
  }

  boot_disk {
    disk_id = yandex_compute_disk.boot_disk.id
  }

  network_interface {
    subnet_id = yandex_vpc_subnet.private.id
    nat       = true
    nat_ip_address = yandex_vpc_address.addr.external_ipv4_address[0].address
  }
}

```

```

provisioner "local-exec" {
    command = "echo \"\${self.network_interface.0.nat_ip_address}
ansible_ssh_private_key_file=/home/vboxuser/.ssh/id_ed25519 ansible_user=admin
\n \" > hosts2"
}
metadata = {
    user-data = "${file("./meta.yml")}"
}
}

resource "yandex_container_registry" "my-registry" { # создаем Container
Registry
    name      = "reg"
}

output "public_ip" {
    value = yandex_compute_instance.this.network_interface.0.nat_ip_address
}

```

Применим конфигурацию

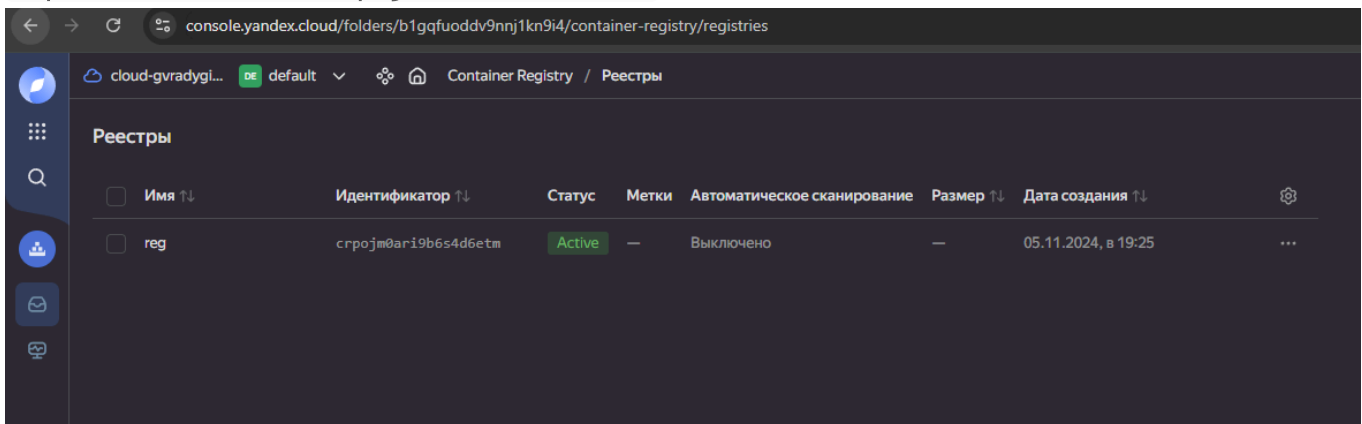
```
terraform apply
```

Аутентифицируемся в registry

```
echo <OAuth-токен> | docker login --username oauth --password-stdin cr.yandex
```

Запишем идентификатор registry в переменную

```
export REGISTRY_ID=crpojm0ari9b6s4d6etm
```



Соберем образ

```
docker build . -t cr.yandex/${REGISTRY_ID}/micro:latest
```

Пушим в registry

```
docker push cr.yandex/${REGISTRY_ID}/micro:latest
```

#### 14. Создадим новый плейбук ycr.yml

```
nano ycr.yml
```

```
- hosts: all
  become: yes
  vars:
    registry_id: "crpnm06nckfkas4ss6k"
  roles:
    - micro-service-from-registry # вызов роли
```

#### 15. Сделаем новую роль

```
mkdir -p ./roles/micro-service-from-registry/tasks
```

```
nano ./roles/micro-service-from-registry/tasks/main.tf
```

```
- name: Docker login # аутентификация в Container Registry
  shell: echo y0_AgAEA7qkn1-EAATuwQAAAAEXP_5TAADqXAsU0a1E6q7gGqYcFL5a38WT0g |
docker login --username oauth --password-stdin cr.yandex
  register: variable_command_output

- name: Docker pull # скачиваем образ из Container Registry
  docker_image:
    name: "cr.yandex/{{ registry_id }}/micro:latest"
    source: pull

- name: Run Docker container # запускаем контейнер на 8080 порту
  docker_container:
    name: micro-service
    image: "cr.yandex/{{ registry_id }}/micro:latest"
    state: started
    ports:
      - "8080:8080"
    restart_policy: always
```

#### 16. Проверим, что все правильно работает

```
sudo ansible-playbook -i hosts2 -vvv ycr.yml
```

```
← → ↻ ⚠ Не защищено 51.250.64.42:8080

# HELP requests_total Total HTTP Requests
# TYPE requests_total counter
requests_total 1.0
# HELP requests_created Total HTTP Requests
# TYPE requests_created gauge
requests_created 1.7308289698279724e+09
host_type container
```

Каталог проекта:

```
}vboxuser@ubuntu7:~/micro$ ls -la
total 92
drwxrwxr-x  5 vboxuser vboxuser 4096 Nov  5 18:15 .
drwxr-x--- 33 vboxuser vboxuser 4096 Nov  5 18:10 ..
-rw-rw-r--  1 vboxuser vboxuser  185 Nov  4 16:44 ansible.cfg
-rw-rw-r--  1 vboxuser vboxuser   63 Nov  4 21:03 container_playbook.yml
-rw-rw-r--  1 vboxuser vboxuser  312 Nov  4 20:27 Dockerfile
-rw-rw-r--  1 vboxuser vboxuser  100 Nov  5 13:32 hosts
-rw-rw-r--  1 vboxuser vboxuser   95 Nov  5 17:26 hosts2
-rw-----  1 vboxuser vboxuser 2491 Nov  4 22:58 key.json
-rw-rw-r--  1 vboxuser vboxuser  694 Nov  4 16:27 main.py
-rw-rw-r--  1 vboxuser vboxuser 1593 Nov  5 16:25 main.tf
-rw-rw-r--  1 vboxuser vboxuser  656 Nov  5 13:22 meta.yml
drwxrwxr-x  5 vboxuser vboxuser 4096 Nov  4 16:04 my_venv
-rw-rw-r--  1 vboxuser vboxuser 1078 Nov  4 17:55 playbook.yml
-rw-rw-r--  1 vboxuser vboxuser  252 Nov  5 00:03 provider.tf
-rw-rw-r--  1 vboxuser vboxuser   24 Nov  4 20:26 requirements.txt
drwxrwxr-x  4 vboxuser vboxuser 4096 Nov  5 17:01 roles
drwxr-xr-x  3 vboxuser vboxuser 4096 Nov  5 00:16 .terraform
-rw-r--r--  1 vboxuser vboxuser  259 Nov  5 00:16 .terraform.lock.hcl
-rw-rw-r--  1 vboxuser vboxuser  182 Nov  5 18:15 terraform.tfstate
-rw-rw-r--  1 vboxuser vboxuser 10336 Nov  5 18:15 terraform.tfstate.backup
-rw-rw-r--  1 vboxuser vboxuser   118 Nov  5 17:46 ycr.yml
```