2. Контейнеризация и виртуализация

Задача 2.1: Управление виртуальной машиной с использованием Vagrant

Vagrant позволяет автоматизировать запуск виртуальных машин.

1. Устанавливаем программы Vagrant и VirtualBox
2. В папке создаем Vagrant файл с помощью команды vagrant init
3. Конфигурируем Vagrant файл согласно нашим требованиям:

```
# The "2" in Vagrant.configure configures the configuration version
Vagrant.configure("2") do |config|


  # Every Vagrant development environment requires a box
  # We choosing the box "bento/ubuntu-20.04"
  config.vm.box = "bento/ubuntu-20.04"
  config.vm.hostname = "vm1"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  config.vm.network "public_network", ip: "192.168.1.2"

  # Customize necessary amount of memory and cpu on the VM:
  config.vm.provider "virtualbox" do |vb|
          vb.memory = "1024"
    vb.cpus = 2
  end

end

```

4. Запускаем виртуальную машину командой vagrant up (по умолчанию выбирается единственная машина, если множество нужно указать имя)

```
PS C:\Users\batyrkhan\vagrant_getting_started> vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'bento/ubuntu-20.04'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'bento/ubuntu-20.04' version '202309.09.0' is up to date...
==> default: Setting the name of the VM: vagrant_getting_started_default_1700802492572_89927
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: bridged
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
```

5. Заходим в vm используя команду vagrant ssh

```
PS C:\Users\batyrkhan\vagrant_getting_started> vagrant ssh
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-162-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Fri 24 Nov 2023 05:10:57 AM UTC

  System load:  0.57               Processes:             161
  Usage of /:   11.7% of 30.34GB   Users logged in:       0
  Memory usage: 23%                IPv4 address for eth0: 10.0.2.15
  Swap usage:   0%                 IPv4 address for eth1: 192.168.1.2


This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento
vagrant@vm1:~$
```

6. Приостанавливаем vm командой vagrant halt и удаляем командой vagrant destroy

```
PS C:\Users\batyrkhan\vagrant_getting_started> vagrant halt
==> default: Attempting graceful shutdown of VM...
```

```
PS C:\Users\batyrkhan\vagrant_getting_started> vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Destroying VM and associated drives...
```

В данном задании мы ознакомились с основные инструкциями по работе с программой Vagrant. По вызовам команд проблемы не было, однако с конфигурирование Vagrant файла необходимо было немного разобраться.

Задача 2.2: Создание многокомпонентного окружения с использованием Vagrant и VirtualBox

1. Создаем в новой папке Vagrant файл командой vagrant init
2. В файле Vagrant конфигурируем две разные виртуальных машин. Настроили разные образы, количество CPU, объем оперативной памяти и конфигурацию сети чтобы обе vm взаимодействовали друг с другом и также имели доступ к интернету через NAT. Также добавили bash script чтобы установил git для каждой vm при их запуске

```ruby
Vagrant.configure("2") do |config|

  # VM1 configuration
  config.vm.define "VM1" do |vm1|
    vm1.vm.box = "bento/ubuntu-20.04"
    vm1.vm.network "private_network", ip: "192.168.33.10"
    vm1.vm.provider "virtualbox" do |vb1|
      vb1.memory = 1024
      vb1.cpus = 2
    end

  # NAT configuration
  vm1.vm.network "forwarded_port", guest: 80, host:8081, auto_correct: true

  # Provisioning bash script
  vm1.vm.provision "shell", path: "vm1_setup.sh"
  end

  # VM2 configuration
  config.vm.define "VM2" do |vm2|
    vm2.vm.box = "ubuntu/focal64"
    vm2.vm.network "private_network", ip: "192.168.33.11"
    vm2.vm.provider "virtualbox" do |vb2|
      vb2.memory = 2048
      vb2.cpus = 1
    end
  vm2.vm.network "forwarded_port", guest: 80, host:8082, auto_correct: true
  vm2.vm.provision "shell", path: "vm2_setup.sh"
  end

end
```

3. Поднимаем vm командой vagrant up

```
PS C:\Users\batyrkhan\vagrant_multiple_environments> vagrant up
Bringing machine 'VM1' up with 'virtualbox' provider...
Bringing machine 'VM2' up with 'virtualbox' provider...
==> VM1: Checking if box 'bento/ubuntu-20.04' version '202309.09.0' is up to date...
==> VM1: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> VM1: flag to force provisioning. Provisioners marked to run always will still run.
==> VM2: Checking if box 'ubuntu/focal64' version '20231011.0.0' is up to date...
==> VM2: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> VM2: flag to force provisioning. Provisioners marked to run always will still run.
```

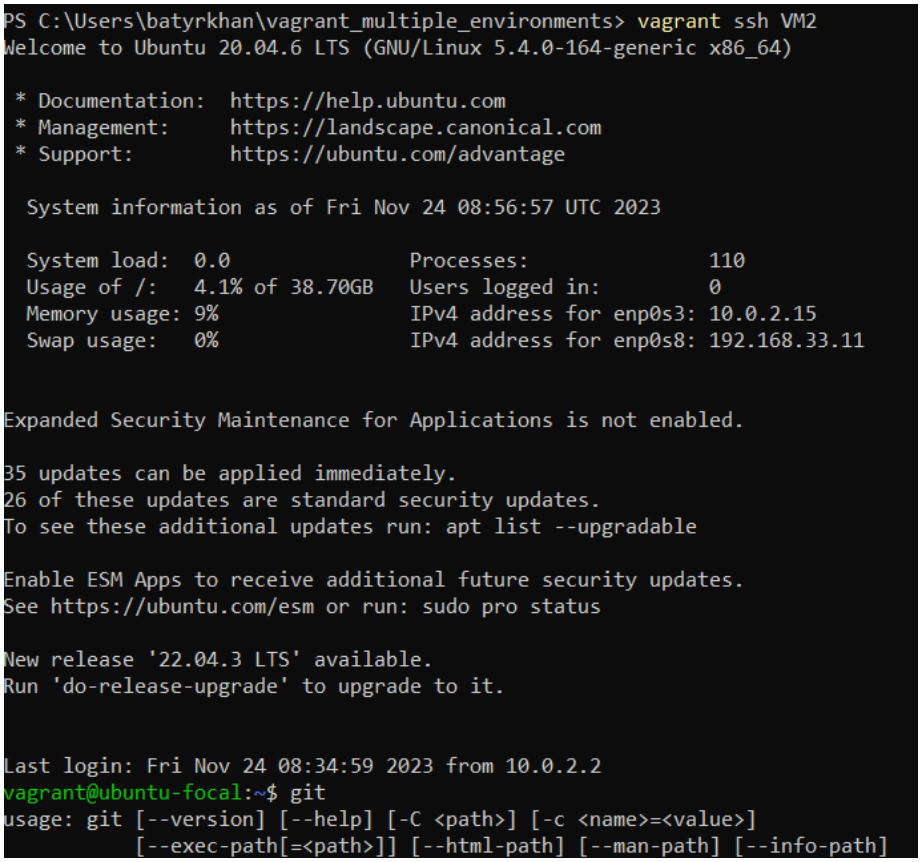
4. Подключаемся к первой VM1 командой vagrant ssh VM1

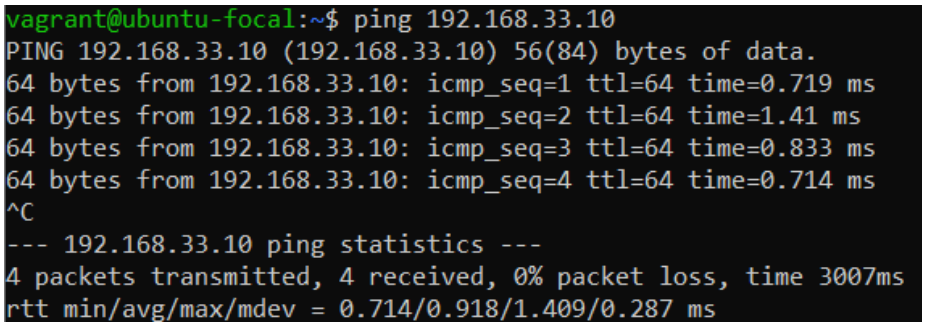
```
PS C:\Users\batyrkhan\vagrant_multiple_environments> vagrant ssh VM1
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-162-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Fri 24 Nov 2023 08:51:03 AM UTC

  System load:  0.0               Processes:             142
  Usage of /:   12.4% of 30.34GB  Users logged in:       0
  Memory usage: 21%               IPv4 address for eth0: 10.0.2.15
  Swap usage:   0%                IPv4 address for eth1: 192.168.33.10


This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento
Last login: Fri Nov 24 08:32:17 2023 from 10.0.2.2
vagrant@vagrant:~$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
```

5. Пробуем пинговать со второй VM2

```
vagrant@vagrant:~$ ping 192.168.33.11
PING 192.168.33.11 (192.168.33.11) 56(84) bytes of data.
64 bytes from 192.168.33.11: icmp_seq=1 ttl=64 time=1.33 ms
64 bytes from 192.168.33.11: icmp_seq=2 ttl=64 time=1.51 ms
64 bytes from 192.168.33.11: icmp_seq=3 ttl=64 time=0.822 ms
64 bytes from 192.168.33.11: icmp_seq=4 ttl=64 time=0.984 ms
64 bytes from 192.168.33.11: icmp_seq=5 ttl=64 time=1.30 ms
64 bytes from 192.168.33.11: icmp_seq=6 ttl=64 time=0.757 ms
64 bytes from 192.168.33.11: icmp_seq=7 ttl=64 time=0.963 ms
64 bytes from 192.168.33.11: icmp_seq=8 ttl=64 time=1.10 ms
64 bytes from 192.168.33.11: icmp_seq=9 ttl=64 time=0.902 ms
^C
--- 192.168.33.11 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8030ms
rtt min/avg/max/mdev = 0.757/1.074/1.509/0.240 ms
```

6. Таким же образом проверяем вторую VM2





Данное задание демонстрирует как можно легко создавать множество виртуальных машин используя Vagrant. Самой главной задачей является правильное конфигурирование Vagrant файла. В процессе конфигурации сетей для двух vm происходили ошибки с распределением ip адресов. Необходимо было указать разные подсети для vm чтобы не сталкивались с другими устройствами сети, поэтому ip адрес был изменен с 192.168.1.3 на 192.168.33.10.

Задача 2.3: Автоматизация конфигурации и администрирование виртуальной машины с помощью Ansible

Задание в процесс работы. Должен закончить до защиты проектов.

Задача 2.4: Создание и запуск контейнера с веб-приложением в Docker

Docker позволяет запускать приложения с помощью контейнеров

1. Сперва устанавливаем Docker на локальной машине
2. Далее создаем простое веб-приложение (Например: to _do_list)
3. Для создания контейнеров сперва создаем Dockerfile где размещаем все необходимые инструкции

```
# syntax=docker/dockerfile:1

FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

В качестве базового образа используем node:18-alpine (как в примере с docker документации)
- yarn для установки зависимости
- CMD [“node”] для запуска
- используем порт 3000

4. На основе Dockerfile создаем образ используя docker build -t to_do_list .

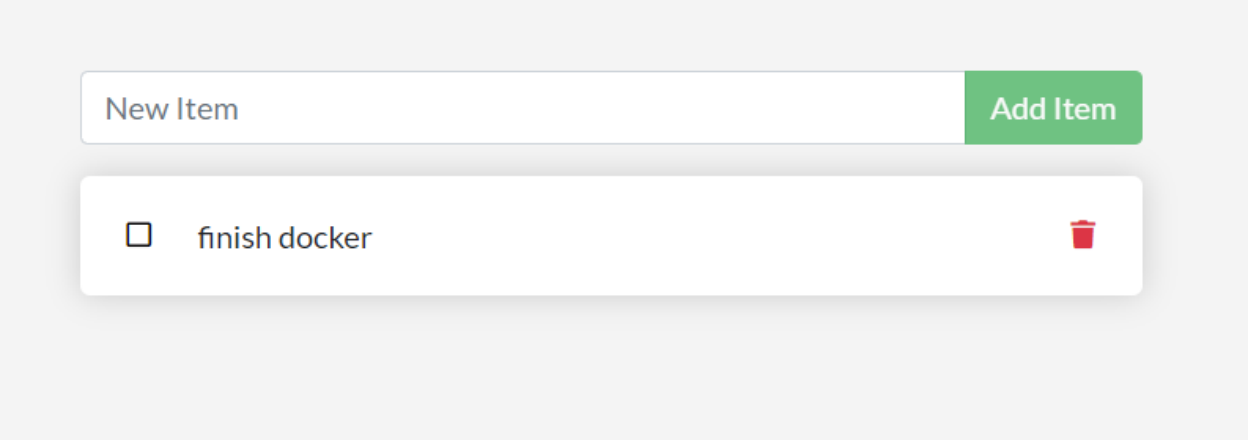
```
PS C:\Users\batyrkhan\Desktop\Main\Itransition_files\docker_tutorial\docker_app> docker build -t to_do_list .
[+] Building 45.7s (13/13) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 32B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => resolve image config for docker.io/docker/dockerfile:1
```

```
PS C:\Users\batyrkhan\Desktop\Main\Itransition_files\docker_tutorial\docker_app> docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
to_do_list   latest    39214aa04ac0   4 minutes ago   223MB
PS C:\Users\batyrkhan\Desktop\Main\Itransition_files\docker_tutorial\docker_app>
```

5. Как только все необходимые образы скачались на локальную машину и вся инструкция в Dockerfile завершилась вызываем контейнер командой docker run -dp 127.0.0.1:3000:3000 to_do_list

```
PS C:\Users\batyrkhan\Desktop\Main\Itransition_files\docker_tutorial\docker_app>  docker run -dp 127.0.0.1:3000:3000 to_do_list
39c18e8db6a764a0889cd02471b1a88738121f606ac719d2e43f01d369fdbccf
PS C:\Users\batyrkhan\Desktop\Main\Itransition_files\docker_tutorial\docker_app> docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED             STATUS             PORTS                     NAMES
39c18e8db6a7   to_do_list   "docker-entrypoint.s…"   About a minute ago   Up About a minute   127.0.0.1:3000->3000/tcp   silly_galois
```

6. Когда контейнер создался можем переходить на само веб-приложение по ссылке http://localhost:3000/



В данной задаче мы ознакомились с основами Docker для чего необходим Docker, как создать инструкцию в Dockerfile, как создается образ и запускается контейнер. В итоге получаем наше веб-приложение который должен работать безупречно на различных машинах.

Задача 2.5: Docker Compose

Задание в процесс работы. Должен закончить до защиты проектов.

Задача 2.6: Знакомство с Kubernetes и Minikube

Kubernetes позволяет оркестровать множество контейнеров.

1. Сначала устанавливаем Minikube для создания мини-кластеров Kubernetes на локальной машине. Minikube автоматический устанавливает kubectl
2. Запускаем локальный мини-кластер Kubernetes используя команду minikube start

```
PS C:\minikube> minikube start
* minikube v1.32.0 on Microsoft Windows 11 Pro 10.0.22621.2715 Build 22621.2715
* Using the virtualbox driver based on existing profile
* Starting control plane node minikube in cluster minikube
* Restarting existing virtualbox VM for "minikube" ...
```

3. С помощью команды kubectl cluster-info проверяем состояние кластера Kubernetes и доступность API сервера

```
PS C:\minikube> kubectl cluster-info
Kubernetes control plane is running at https://192.168.59.100:8443
CoreDNS is running at https://192.168.59.100:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

4. Для конфигурирования подов создаем yaml файл, где в качестве контейнера выбираем веб сервер nginx. Далее запускаем поду в кластере командой kubectl create -f my-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

```
PS C:\minikube> kubectl create -f my-pod.yaml
pod/nginx created
```

5. Чтобы узнать состояния пода и сервиса воспользуемся командами kubectl get pods и kubectl get services

```
PS C:\minikube> kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          3m28s
PS C:\minikube> kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   31h
```

Также для получения большей информации о поде воспользуемся командой kubectl describe pod

```
PS C:\minikube> kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.59.100
Start Time:       Fri, 24 Nov 2023 12:00:27 +0600
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.0.5
```

6. Команда kubectl logs нужна для просмотра журналов пода, а для того чтобы зайти внутрь пода и делать отладку мы воспользуемся командой kubectl exec -it nginx –/bin/bash

```
PS C:\minikube> kubectl logs nginx
PS C:\minikube> kubectl exec -it nginx -- /bin/bash
root@nginx:/#
```

7. Останавливаем и удаляем под и сервис командами kubectl delete pod и kubectl delete service соответственно

```
PS C:\minikube> kubectl delete pod nginx
pod "nginx" deleted
```

```
PS C:\minikube> kubectl delete service kubernetes
service "kubernetes" deleted
```

8. В конце останавливаем кластер с помощью команды minikube stop

```
PS C:\minikube> minikube stop
* Stopping node "minikube"  ...
* 1 node stopped.
```

В данном задании мы рассмотрели возможность Kubernetes управлять контейнером, процесс конфигурации кластера и работу с подами. Хотя мы рассмотрели маленький пример с использованием одного контейнера, возможности Kubernetes по настоящему проявятся при оркестрировании множества контейнеров на различных подах.