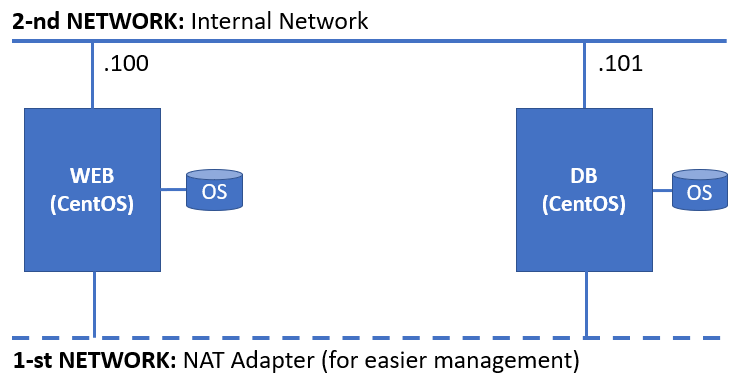
# Practice M1: Introduction to DevOps

The lab environment has the following structure:



The following tasks are executed on different machines. It is stated clearly on which machine a task is being executed

VM templates can be downloaded from this URL: <https://zahariev.pro/go/dob-templates>

Do not hurry to download or install anything, wait until you reach the appropriate section

## Part 1: Introduction

Nothing to do here, just talking/discussing 😊

## Part 2: Manual approach

### Preparation

We must ensure that we have a working and supported hypervisor installed

**VirtualBox** is the preferred option as is it both free and well supported by **Vagrant**

Until the end of this lab we will be working with VirtualBox

#### Install VirtualBox (on a Windows host)

Please make sure that no other virtualization solutions are installed and active

If there are any, please uninstall them

Navigate to the download page (<https://www.virtualbox.org/wiki/Downloads>) and download the latest version

Make sure that you have administrative access and double-click on the downloaded file to initiate the installation process

As you will see it is pretty straight-forward

#### Install VirtualBox (on a CentOS host)

The steps are more or less the same no matter which **Linux** distribution is installed on the host

* Open a terminal session

**wget http://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo**

**sudo mv virtualbox.repo /etc/yum.repos.d/**

**sudo dnf upgrade -y**

**sudo dnf install -y gcc make kernel-headers kernel-devel**

**sudo dnf install -y VirtualBox-6.1**

**sudo usermod -a -G vboxusers $USER**

* Log-out and log-on again
* Check that **VirtualBox** is correctly installed
* Should you see any errors when trying to start a VM, you may need to run

**sudo /sbin/vboxconfig**

* If the above end with an error, check the **/var/log/vbox-setup.log** file
* Most probably you will have to install additional packages, for example

**sudo dnf install elfutils-libelf-devel**

* And re-run the **/sbin/vboxconfig** command

### Application Deployment

Now, that we have the infrastructure set up, we can continue with the application deployment steps

Let’s imagine that we have to deploy a two-tier application with web-based (PHP) front-end and MariaDB/MySQL used for database

Following established practices we will deploy the two components on two different machines (you can refer to the picture in the beginning of the document)

#### Import two templates (on the host)

For the rest of the lab we can use either one of the three available templates – **CentOS**, **openSUSE**, or **Ubuntu**

Let’s use the **CentOS** one (should you want to experiment with another one, be sure to adjust the steps)

* Start VirtualBox
* Import **centos.ova** as **WEB** machine
* Import **centos.ova** as **DB** machine
* Set networking mode for the first (and only) network adapter to **NAT \***
* Add a second network adapter to both virtual machines and set it to Internal Network mode
* Don’t forget to set static address on second network adapters and test the connectivity there
* Start both machines

*\* For easier interaction you can set up a port forwarding rules to port 22 of both virtual machines*

#### Install software (on WEB VM)

* Log-on to WEB with **vagrant** user and **vagrant** for password
* The **root** password is the same (**vagrant**)
* Change the hostname to **web** by executing

**sudo hostnamectl set-hostname web**

* Log-out and log-on again
* Install **Apache**, **PHP**, **Git**, and other required/related packages

**sudo dnf install -y httpd php php-mysqlnd git**

* Configure **Apache** to start automatically and start it

**sudo systemctl enable httpd**

**sudo systemctl start httpd**

* Check that we have connectivity \*
* We can adjust the firewall in one of the following ways:
  + Stop and disable it

**sudo systemctl stop firewalld**

**sudo systemctl disable firewalld**

* + Or open the appropriate port or service

**sudo firewall-cmd --add-service=http --permanent**

**sudo firewall-cmd --reload**

* We can check again

*\* Depending on the network mode of the virtual machines, you may need to set up a port forwarding rule*

#### Deploy the application (on WEB VM)

* Go to home folder
* Execute **git clone** [**https://github.com/shekeriev/dob-module-1**](https://github.com/shekeriev/dob-module-1)
* Copy all files from **dob-module-1/web** to **/var/www/html**
* Try to open the page on the host \*

*\* Depending on the network mode of the virtual machines, you may need to set up a port forwarding rule*

#### Install software (on DB VM)

* Log-on to DB with **vagrant** for user and **vagrant** for password
* The **root** password is the same (**vagrant**)
* Change the hostname to **db** by executing

**sudo hostnamectl set-hostname db**

* Log-out and log-on again
* Install **MariaDB** client and server components

**sudo dnf install -y mariadb mariadb-server git**

* Enable and start the service

**sudo systemctl enable mariadb**

**sudo systemctl start mariadb**

* Do some initial configuration

**sudo myql\_secure\_installation**

#### Configure database (on DB VM)

* Go to home folder
* Execute **git clone** [**https://github.com/shekeriev/dob-module-1**](https://github.com/shekeriev/dob-module-1)
* Navigate to the **db** folder
* Examine the SQL script
* Execute the script against the database

**mysql -u root -p < db\_setup.sql**

* Log-on and check that the data is there
* Modify the firewall state:
  + Disable it:

**sudo systemctl stop firewalld**

**sudo systemctl disable firewalld**

* + Or open the appropriate port:

**sudo firewall-cmd --add-port=3306/tcp --permanent**

**sudo firewall-cmd --reload**

#### Configure and test the application (on WEB VM)

* Check the **config.php** script if there is a need to change connection parameters
* Open browser and check the final result
* In case of connectivity error execute

**sudo setsebool -P httpd\_can\_network\_connect=1**

* Other option is to modify **SELinux** mode (**/etc/sysconfig/selinux**)

## Part 3: Vagrant

### Preparation

In order to automate the process we saw so far, we will need a helper tool

Vagrant is a good fit for this

To install it, we must navigate to here: <https://www.vagrantup.com/downloads>

Should you need an earlier version, you can check here: <https://releases.hashicorp.com/vagrant/>

#### Install Vagrant (on Windows host)

Make sure that you have administrative access and double-click on the downloaded file to initiate the installation process

As you will see it is pretty straight-forward

#### Install Vagrant (on CentOS host)

Let us install the **Vagrant** tool which will help us automate our infrastructure creation and management to a good extent

The steps are more or less the same no matter which **Linux** distribution is installed on the host

* First, we download the package locally

**wget https://releases.hashicorp.com/vagrant/2.2.15/vagrant\_2.2.15\_x86\_64.rpm**

* Then we install it

**sudo rpm -ivh vagrant\_2.2.15\_x86\_64.rpm**

* Then we check that it is working

**vagrant version**

* We can add bash completion should we want it. Download a 3rd party completion file

**wget https://github.com/hashicorp/vagrant/raw/master/contrib/bash/completion.sh -O vagrant-completion.sh**

* Move (install) the completion script

**sudo mv vagrant-completion.sh /etc/bash\_completion.d/**

#### Create Vagrant box (on host)

We can create our own boxes or use ones already created by others

The basic steps to create a box of our own is to create a simple VM with minimal parameters:

* 1 CPU
* 1 GB RAM
* 48 GB HDD (Dynamic)
* No Audio
* 1 NIC in NAT mode (with a port forwarding rule SSH/TCP/2222/22)

Then we must install the OS (CentOS 8 in our case) again with minimal (or what we find suitable) profile:

* Turn off the **kdump** utility
* Enable the network and set a hostname
* Set the appropriate time zone
* Select the OS profile – usually **Minimal**

Once, we have the machine up and running, we must log on and do some final adjustments:

* Make sure that there is SSH installed and running
* Should we want, we can disable services like the firewall
* Upgrade all installed packages

**sudo dnf upgrade**

* Add new packages if you see fit. For example, install at least these:

**sudo dnf install kernel-devel gcc make tar bzip2 wget elfutils-libelf-devel**

* Mount the VirtualBox Add-ons media and install them

**sudo mount /dev/sr0 /mnt**

**sudo /mnt/VBoxLinuxAdditions.run**

* Add the **vagrant** user to the **vboxsf** group

**sudo usermod -aG vboxsf vagrant**

* Set the GRUB waiting time to **0**

**sudo vi /etc/default/grub**

**sudo grub2-mkconfig -o /boot/grub2/grub.cfg**

* Add the **vagrant** user to the **sudoers** list and allow it to **sudo** without entering password

**echo "vagrant ALL=(ALL) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/vagrant**

* Install the **vagrant** insecure key

**mkdir -m 0700 -p /home/vagrant/.ssh**

**wget --no-check-certificate \**

**https://raw.github.com/mitchellh/vagrant/master/keys/vagrant.pub \**

**-O /home/vagrant/.ssh/authorized\_keys**

**chmod 0600 /home/vagrant/.ssh/authorized\_keys**

* Clean up the DNF cache

**sudo dnf clean all**

* Make sure that the hard disk is aligned

**sudo dd if=/dev/zero of=/EMPTY bs=1M status=progress**

**sudo rm -f /EMPTY**

* Reboot the VM

Now, we are ready to build our box

* Create a folder to host the box
* If we are on a **Linux** host, we can execute the following

**mkdir -p /home/$USER/Vagrant/centos && cd /home/$USER/Vagrant/centos**

* While the VM is still running, build the box

**vagrant package --base <VM Name>**

Let us test it

* Add the box to the local catalog

**vagrant box add <Box Name> package.box**

* Create configuration base on the local box

**vagrant init <Box Name>**

* Power on the machine

**vagrant up**

* Connect to the machine

**vagrant ssh**

* Explore what's inside the machine
* Close the SSH session

**exit**

* Turn off and delete the machine

**vagrant destroy --force**

Finally, we can publish our box to **Vagrant Cloud**

* Navigate to **https://app.vagrantup.com** and login
* Click on **New Vagrant Box**
* Enter the details and click on **Create** **box**
* Enter version and description and click **Create** **version**
* Click on **Add a provider**
* For provider set **virtualbox** and click on **Continue to upload**
* Click on the **Browse** button and navigate to the box file
* Once the upload is complete, click on the **Update provider** button
* Click on the box name
* Click on **Release...**
* Click on **Release version**

Now, we can continue either with our own box or the cited in the next steps

#### First run of the box (on host)

* Create a new work folder **vagrant/dob-m1-1**
* Go there
* Execute (substitute **shekeriev** with your own repository name)

**vagrant init shekeriev/centos-8-minimal**

* Examine the created **Vagrantfile**
* Power-on the machine

**vagrant up**

* Start ssh session to the machine

**vagrant ssh**

* Browse the machine
* Exit the session
* List all local machines

**vagrant global-status**

* List all boxes

**vagrant box list**

* Destroy the machine

**vagrant destroy --force**

### Application Deployment

For the next two tasks we will need the archive file available on the site of the current module

Download it and extract it somewhere

#### Automation Level 1 (on host)

Let’s assume that the files are extracted to a folder **~/DOB/M1/**

* Navigate to the folder **3-1**
* Examine the **Vagrantfile**
* Modify the settings if needed
* Check the sub-folders
* Execute

**vagrant up**

* Open a browser and test the application
* Destroy the machines

**vagrant destroy --force**

#### Alternative approach #1 (on host)

Let’s assume that the files are extracted to a folder **~/DOB/M1/**

* Navigate to the folder **3-2**
* Examine the **Vagrantfile**
* Modify the settings if needed
* Check the sub-folders
* Execute

**vagrant up**

* Open a browser and test the application
* Destroy the machines

**vagrant destroy --force**

#### Alternative approach #2 (on host)

Can you think of another alternative approach?

For example, to use **git clone** within the **Vagrantfile** during the machine provision phase

Try to do it 😉