# Practice M1: Introduction to Docker

For this practice we will use an infrastructure like this:

Graphical user interface, application

Description automatically generated

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/devops-vm-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: Basic Toolkit

### Preparation

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

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

Of course, other virtualization solutions like **VMware Workstation** and **Hyper-V** can do the job as well but will require some tweaks

Be sure to fill the **Introduction Form** ([*https://zahariev.pro/go/devops-about-you*](https://zahariev.pro/go/devops-about-you)*) a*nd state at least your configuration – hardware, OS and virtualization solution. This will help to adapt the content to better fit the majority

Until the end of this practice, we will be working with **VirtualBox** on **Intel-compatible** hardware (mostly)

*Please make sure that no other virtualization solutions are installed and active (this includes also Docker Desktop as it relies on an existing hypervisor)*

*If there are any, please uninstall them (or just go with them)*

*If you already have VirtualBox installed, skip the following section*

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

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 Red Hat-compatible host)

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

*In any case, check the official documentation:* [*https://www.virtualbox.org/wiki/Linux\_Downloads*](https://www.virtualbox.org/wiki/Linux_Downloads)

Open a terminal session

Download the repository configuration file

**wget https://download.virtualbox.org/virtualbox/rpm/el/virtualbox.repo**

*Even though Fedora is part of the same family, for it the above should become*

***wget https://download.virtualbox.org/virtualbox/rpm/fedora/virtualbox.repo***

*So, as advised above, check the official documentation 😉*

Move it to the appropriate target folder

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

Update currently installed packages

**sudo dnf upgrade -y**

Install any prerequisite packages that may be missing

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

Install the **VirtualBox** package

**sudo dnf install -y VirtualBox-7.1**

Add the current user to the appropriate user group

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

Log-out and log-on again

Check that **VirtualBox** is correctly installed and working

*Should you see any errors when trying to start a VM, you may need to run*

***sudo /sbin/vboxconfig***

*If the above ends 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*

*The actual steps vary depending on the distribution in use and if the system is BIOS or UEFI-based*

### Manual 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 at the beginning of the document)

#### Import a template (on the host)

*We will be using a template to save some time. Of course, should we want, we can start fresh and install the OS in the virtual machines manually*

For the rest of the lab, we can use either one of the available templates – **AlmaLinux OS**, **Debian**, or **openSUSE**

Let’s use the **AlmaLinux OS** one (in fact, it is a **AlmaLinux OS 9.6** template). Should you want to experiment with another one, be sure to adjust the steps

Start **VirtualBox**

Import **almalinuxos-9.6.ova** as **WEB** machine

Import **almalinuxos-9.6.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. Make sure that the name of the internal network is the same for both machines

Start both machines

*\* For easier interaction you can set up a port forwarding rules to port* ***22*** *of both virtual machines. For example, for the* ***WEB*** *machine redirect* ***22*** *to* ***10001*** *on the host and for the* ***DB*** *machine redirect* ***22*** *to* ***10002*** *on the host*

*\*\* Don’t forget to set static address on second network adapters and test the connectivity there. Use for example the* ***192.168.99.0/24*** *network*

#### Install software (on WEB VM)

Log-on to the **WEB** machine with **vagrant** user and **vagrant** for password

*Note: The* ***root*** *password is the same (****vagrant****)*

Set the static IP address on the second network interface. For example, **192.168.99.100/24**

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 on boot and start it

**sudo systemctl enable httpd**

**sudo systemctl start httpd**

Check that we have connectivity both locally and remotely \*

We can adjust the firewall in one of the following two 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. For example, port* ***80*** *on the* ***VM*** *to port* ***8080*** *on the host*

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

Go to the home folder of the **vagrant** user

Execute the following to clone the existing repository and get the application’s files locally

**git clone https://github.com/shekeriev/bgapp**

Copy all files from **bgapp/web** to **/var/www/html**

**sudo cp -v bgapp/web/\* /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. For example, port* ***80*** *on the* ***VM*** *to port* ***8080*** *on the host*

#### Install software (on DB VM)

Log-on to **DB** machine with **vagrant** for user and **vagrant** for password

*Note: The* ***root*** *password is the same (****vagrant****)*

Set the static IP address on the second network interface. For example, **192.168.99.101/24**

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 mysql\_secure\_installation**

*Note: Depending on the circumstances, the above command may have to become* ***sudo mariadb-secure-installation***

#### Configure database (on DB VM)

Go to home folder of the **vagrant** user

Execute the following to clone the existing repository and get the application’s files locally

**git clone https://github.com/shekeriev/bgapp**

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

**mysql -u root -p**

**use bulgaria;**

**select \* from cities;**

**quit**

Modify the firewall state in either of the following two ways

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. It should be working

In case of connectivity error execute the following on the web server to check where the issue is

**curl http://localhost**

**php /var/www/html/index.php**

**ping db**

Under **RedHat**-based distributions, we will see that in one of the attempts we can see the data and in the other – we cannot

This is caused by the **SELinux** suite, and we can tackle the issue by executing the following

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

Other option is to modify **SELinux** mode (**/etc/sysconfig/selinux**) to either **permissive** or **disabled**

Last, but not least, the issue may lie in a non-working name resolution between the machines. If so, adjust the **/etc/hosts** file accordingly

### Automated Application Deployment

#### Preparation

In order to automate the process, we have seen 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 Red Hat-compatible 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/2.4.6/vagrant-2.4.6-1.x86\_64.rpm**

Then we install it

**sudo rpm -ivh vagrant-2.4.6-1.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 *(you may need to install the* ***bash-completion*** *package as well)*

**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. We could use the following:

* 1 CPU
* 2 GB RAM
* 32 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 (**AlmaLinux OS 9.6** in our case) again with a 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**
* Add the **vagrant** user and set its password to **vagrant**
* Enable the **root** user and use the same password

For the purpose of the demo, we will reuse one of the available templates

Let’s use again **AlmaLinux OS**. The template is called **almalinuxos-9.6.ova** *(it is in fact* ***AlmaLinux OS 9.6****)*

Import a **VM** from the template to save some time *(don’t forget the port forwarding rule)*

Once we have the machine up and running, we must log on as the **root** user 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

**dnf upgrade**

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

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

Mount *(****Devices > Insert Guest Additions image****)* the **VirtualBox** **Guest Additions** media and install them

**mount /dev/sr0 /mnt**

**/mnt/VBoxLinuxAdditions.run**

*Or execute* ***/mnt/VBoxLinuxAdditions-arm64.run*** *for ARM64-based machine*

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

**usermod -aG vboxsf vagrant**

Set the **GRUB** waiting time to **0** *(this will make our machine boot faster)*

**vi /etc/default/grub**

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

Add the **vagrant** user to the **sudoers** list and allow it to **sudo** without entering password (if not already done in one or another way)

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

Close the session and open a new one as the **vagrant** user

Install the **vagrant** insecure key. First, create the target folder

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

Download the key

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

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

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

Set the permissions of the key

**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**

**sudo reboot**

Do not forget to eject *(****Optical Drives > Remove disk from virtual drive****)* the additions media or any other media that may be still attached to the machine

Now, we are ready to build our box *(execute the next steps* ***on the host and not in the VM****)*

Create a folder to host the box (on the host machine)

If we are on a **Linux** host, we can execute the following to create and enter the work folder

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

Once we are inside the folder, we can start the box creation process

While the **VM** is still running, build the box with *(replace <VM Name> with the name of the VM)*

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

Let’s test it

Add the box to the local catalog *(replace <Box Name> with the name you want to use for your box)*

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

Create a configuration based 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 **HashiCorp Cloud Platform** (or you can skip this step and use either your locally available box or use an already available public box)

Navigate to **https://portal.cloud.hashicorp.com** and login

We will assume that there is a project created already, and it contains a Vagrant Registry

Once inside the registry, click on **Create Box**

Enter the details and click on **Save & continue**

Enter version (for example, 0.1) and description and click **Save & continue**

For provider set **virtualbox**

Change **File hosting** to **Upload file**

Click on the **Choose File** button and navigate to the box file

Set the appropriate architecture

Click the **Save** button

The uploading process will begin

Once the uploading is done, click the **Release now** button

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 **~/do1/m1/test**

Go there and execute *(replace <repo/box-name> with your repository and box name)*

**vagrant init <repo/box-name>**

Instead of your repository and box, you can use for example, either **shekeriev/almalinux-9.6** or **shekeriev/debian-12.11**

Examine the created **Vagrantfile**

Power-on the machine

**vagrant up**

Start **SSH** session to the machine

**vagrant ssh**

Browse the machine by executing a few commands

Exit the session

**exit**

List all local machines

**vagrant global-status**

It should be listed indicating that it is in running state

List all boxes

**vagrant box list**

It should appear on the list together with any other boxes you may have

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 into a folder **~/do1/m1/**

Navigate to the folder **2-1**

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 **~/do/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 😉

## Part 3: Containerization

**References**

<https://docs.docker.com/engine/install/centos/>

<https://docs.docker.com/engine/install/linux-postinstall/>

**Commands**

**version, system info, container run**

### Installation and Configuration (Standard Approach)

#### Preparation

Spin up either an **AlmaLinux OS** or **Debian** based virtual machine

You can do it either by importing a template or using **Vagrant**

Let us go with the second approach, as it is more manageable, and this is what we are aiming for

Open a terminal session and navigate to an empty folder of your choice

Invoke the initialization command to create an empty environment for **AlmaLinux OS**

**vagrant init --minimal shekeriev/almalinux-9.6**

Or for **Debian**

**vagrant init --minimal shekeriev/debian-12.11**

Now open the **Vagrant** file with a text editor of your choice

If you are working in a graphical environment of **Windows**, **Linux**, or **macOS**, you can install the **VSCode** free editor by **Microsoft** and add one of available **Vagrant** plugins *(for example the* ***Vagrantfile Support*** *by* ***Marco Stazi*** *or* ***Vagrant*** *by* ***Baptist Benoist****)*

Now that we are in, let us add one block, between the **config.vm.box** and **end**, to set some port forwarding and change the memory size

**config.vm.network "forwarded\_port", guest: 8080, host: 8080, auto\_correct: true**

**config.vm.provider :virtualbox do |vb|**

**vb.customize ["modifyvm", :id, "--memory", "2048"]**

**end**

Save and close the file

Return to the terminal session and start the environment

**vagrant up**

And then, once the machine is up and running, establish **ssh** session to it

**vagrant ssh**

#### Remove old installations

*This is not directly applicable to our case, as we start fresh without existing* ***Docker*** *installation*

*Should you have an old installation of* ***Docker****, it is recommended to remove it first*

##### Red Hat-based

*Under* ***Red Hat****-based distributions we can execute this*

***sudo dnf remove docker docker-client docker-client-latest \***

***docker-common docker-latest docker-latest-logrotate \***

***docker-logrotate docker-selinux docker-engine-selinux docker-engine***

##### Debian

*Under* ***Debian****-based distributions we can execute this*

***sudo apt-get remove docker docker-engine docker.io containerd runc***

#### Installation preparation

First, we must make sure that we have the appropriate repository registered

##### Red Hat-based

Add the repository

**sudo dnf -y install dnf-plugins-core**

**sudo dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo**

##### Debian

First, refresh package information and install the required packages

**sudo apt-get update**

**sudo apt-get install ca-certificates curl gnupg lsb-release**

Then add the repository key (change **debian** with **ubuntu** if working on **Ubuntu**)

**sudo install -m 0755 -d /etc/apt/keyrings**

**sudo curl -fsSL https://download.docker.com/linux/debian/gpg \**

**-o /etc/apt/keyrings/docker.asc**

**sudo chmod a+r /etc/apt/keyrings/docker.asc**

Finally, add the repository (change **debian** with **ubuntu** if working on **Ubuntu**)

**echo \**

**"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian \**

**$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \**

**sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

#### Docker installation

Now, we are ready to install **Docker** itself

##### Red Hat-based

To install **Docker**, execute the following:

**sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin**

Let us start the **docker** daemon and check its status

**sudo systemctl start docker**

**systemctl status docker**

##### Debian

To install **Docker**, first we must update package information

**sudo apt-get update**

And then install the necessary packages

**sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin**

Finally, check that the service started successfully

**systemctl status docker**

#### Post installation test

If the service is working normally, we can receive more information about the installed versions and the current configuration with

**sudo docker version**

**sudo docker system info**

And let us start our very first container

**sudo docker container run hello-world**

Or this one:

**sudo docker container run shekeriev/welcome-do:latest**

#### Additional settings

To be able to work with **docker** without the need to use always **sudo**, we must add our user to the **docker** group

**sudo usermod -aG docker $USER**

To apply the change, we must log off and then log on back again

We can check by executing any of the previous **docker** commands, for example

**docker container run hello-world**

As a last step, we can mark the **docker** service for automatic start on boot (this is not necessary on **Debian**/**Ubuntu** as it is automatically done)

**sudo systemctl enable docker**

For more post-installation settings like remote connectivity, storage options, etc. visit the following address

<https://docs.docker.com/engine/install/linux-postinstall/>

#### Clean up

Close the session to the VM

And being in the folder where we started (where we have our minimal **Vagrantfile**), execute

**vagrant destroy --force**

This will stop and delete the virtual machine

### Installation and Configuration (Alternative Way)

Besides the way we saw, there are two other ways to install **Docker**

One of them is to download the archive and deal with everything manually

You can check more on this here: <https://docs.docker.com/engine/install/binaries/>

The other way is to use the **Get Docker** script

Is simple. Just download the script

**curl -fsSL https://get.docker.com -o get-docker.sh**

Should you want, you can do a dry run with

**DRY\_RUN=1 sh ./get-docker.sh**

Or do the actual installation with

**sudo sh get-docker.sh**

That is, it. Of course, any post-installation actions are applicable here as well

### Installation and Configuration (The Vagrant Way)

Let us try to write an all-in-one **Vagrantfile** that can be used to spin up a VM with **docker** installed and configured

Create a new folder and switch to it

Open a new **Vagrantfile** for editing

#### Red Hat-based

You can copy and paste what we have in our first **Vagrantfile** and extend it or start clean

Anyway, you should type the following text

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

    config.vm.define "docker" do |docker|

        docker.vm.box="shekeriev/almalinux-9.6"

        docker.vm.hostname = "docker.do1.lab"

        docker.vm.network "private\_network", ip: "192.168.99.100"

        docker.vm.network "forwarded\_port", guest: 8080, host: 8080, auto\_correct: true

        docker.vm.provision "shell", path: "docker-setup.sh"

        docker.vm.provider :virtualbox do |vb|

            vb.customize ["modifyvm", :id, "--memory", "2048"]

      end

    end

  end

Save and close the file

Now create a new **docker-setup.sh** file

Type the following (these are the steps we did manually)

#!/bin/bash

echo "\* Add hosts ..."

echo "192.168.99.100 docker.do1.lab docker" >> /etc/hosts

echo "\* Add Docker repository ..."

dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

echo "\* Install Docker ..."

dnf install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

echo "\* Enable and start Docker ..."

systemctl enable docker

systemctl start docker

echo "\* Firewall - open port 8080 ..."

firewall-cmd --add-port=8080/tcp –permanent

firewall-cmd --reload

echo "\* Add vagrant user to docker group ..."

usermod -aG docker vagrant

Save and close the file

#### Debian

You can copy and paste what we have in our first **Vagrantfile** and extend it or start clean

Anyway, you should type the following text

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

    config.vm.define "docker" do |docker|

        docker.vm.box="shekeriev/debian-12.11"

        docker.vm.hostname = "docker.do1.lab"

        docker.vm.network "private\_network", ip: "192.168.99.200"

        docker.vm.network "forwarded\_port", guest: 8080, host: 8080, auto\_correct: true

        docker.vm.provision "shell", path: "docker-setup.sh"

        docker.vm.provider :virtualbox do |vb|

            vb.customize ["modifyvm", :id, "--memory", "2048"]

      end

    end

  end

Save and close the file

Now create a new **docker-setup.sh** file

Type the following (these are the steps we did manually)

#!/bin/bash

echo "\* Add hosts ..."

echo "192.168.99.200 docker.do1.lab docker" >> /etc/hosts

echo "\* Add any prerequisites ..."

apt-get update

apt-get install -y ca-certificates curl gnupg lsb-release

echo "\* Add Docker repository and key ..."

curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb\_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

echo "\* Install Docker ..."

apt-get update

apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

echo "\* Add vagrant user to docker group ..."

usermod -aG docker vagrant

Save and close the file

#### Use the Vagrantfile

Give it a try with

**vagrant up**

If everything goes according to the plan, then as a result there should be a working **docker** machine

Once, done with the exploration, do not destroy the machine. We will be using it in the next part

### Working with Containers

**References**

<https://hub.docker.com/explore/>

<https://store.docker.com/>

**Commands**

**search**, **image** **pull**, **image** **ls**, **image** **rm**, **container** **run**, **container** **ls**, **container** **rm**, **container** **create**, **container** **start, container attach, container stop, container prune, container export**, **image import**, **image save**, **image load**

#### Working with containers

We could search for all images that contain **ubuntu** in their name

**docker search ubuntu**

The first result in the list is with the biggest rating and it is marked as an official image

We can download it locally with

**docker image pull ubuntu**

In fact, the above command is equivalent to this one

**docker image pull ubuntu:latest**

As a result of the execution of either of the above two commands, we will have the latest version downloaded locally

Should we want a particular version, we can execute

**docker image pull ubuntu:18.04**

List with the available images can be obtained with the execution of the following command

**docker image ls**

Let us start a container out of the **ubuntu:18.04** image

**docker container run -it ubuntu:18.04**

Once, we are in the container, we can execute the following sequence of commands *(just the black portion)*

**root@35ac9218a880:/# ls**

**bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var**

**root@35ac9218a880:/# ps ax**

**PID TTY STAT TIME COMMAND**

**1 pts/0 Ss 0:00 /bin/bash**

**15 pts/0 R+ 0:00 ps ax**

**root@35ac9218a880:/#**

We can do the usual stuff. For example, install a package

Let’s do it by executing the following

**apt-get update**

**apt-get install inetutils-ping**

Now, we can ping, for example the address of **softuni.bg**

**ping softuni.bg**

It is working 😊 Press **Ctrl+C** to interrupt the ping command

Should we want to temporary exit the container without stopping it, we can do it by pressing and holding the **Ctrl** key, then pressing **P** and then **Q** and finally releasing the **Ctrl** key

Alternatively, we can press **Ctrl+P** and then **Ctrl+Q**

Now, we can ask for the list of all working containers

**docker container ls**

The result should look similar to

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**35ac9218a880 ubuntu:18.04 "/bin/bash" 8 minutes ago Up 8 minutes cocky\_fermat**

Should we want to go back to the container, we can execute

**docker container attach 35ac9218a880**

Where the **35ac9218a880** sequence is the container ID (the first column of the **docker container ls** command)

In fact, we can use even shorter string if it is unique amongst the containers on the host

For example, we can use just **35** in this case

Alternatively, we can use the value from the last column - the name of the container (**cocky\_fermat**)

We could terminate the container by executing the **exit** command

**root@35ac9218a880:/# exit**

If we ask now for the containers with

**docker container ls**

We won’s see it, as it not running anymore (we terminated the only process in the container)

In order to see the list of all containers including the stopped ones, we can execute

**docker container ls -a**

We can create a container with custom name and without starting it with

**docker container create -it --name ubuntu-22 ubuntu /bin/bash**

Then, we can start it by executing the following command

**docker container start -ai ubuntu-22**

Let us exit the container without stopping it. We can use the **Ctrl+P** and **Ctrl+Q** key combination

Then, we can stop it with

**docker container stop ubuntu-22**

Let’s start a few more containers but this time they will run in background or detached mode

Start the first one based on the **Alpine** image (it a tiny one) using this command

**docker container run -d --name dummy1 alpine sleep 1d**

Now, start the second but this time based on the **Busybox** image (even tinier) with

**docker container run -d --name dummy2 busybox sleep 1d**

Let’s check how tiny are those images and compare them to the **Ubuntu** based images

**docker image ls**

Wow, they are tiny indeed 😊

Let’s see the list of running containers

**docker container ls**

Imagine that there were many running containers and we need a way to stop them all

We will need a way to extract just their IDs for example as a list and supply it to the stop command

A list of all running containers can be extracted with

**docker container ls -q**

Now, to combine this with the stop command, we can execute the following

**docker container stop $(docker container ls -q)**

What about the stopped containers? Can we remove them somehow? Yes, we can

Let’s ask for their list with

**docker container ls -a**

Particular stopped container or list of containers, we can remove with

**docker container rm ubuntu-22**

All stopped containers can be deleted with

**docker container prune**

Now, the list of the stopped containers should be empty

**docker container ls -a**

What about the images? How can we delete them?

Let’s check again their list with

**docker image ls**

We can remove one or more images with

**docker image rm ubuntu**

And check again

**docker image ls**

Yes, the image is gone

#### Working with different registries

Okay, we saw that **Docker Hub** is the default registry (**docker info**)

But how we can change it at least temporary, just for a single container?

Check this address: <https://hub.zahariev.pro/v2/_catalog>

Two of the images here are suitable for test – **alpine** and **k8s-environ**

Should we want to see what tags are available for the **k8s-environ** image, we can execute

**curl -X GET https://hub.zahariev.pro/v2/k8s-environ/tags/list**

The first is just a local (for the registry) copy of the publicly available **alpine** image

Let’s try the second one

We must execute the following command

**docker container run -d --name other -p 8080:80 hub.zahariev.pro/k8s-environ**

So, we are prefixing the image with the registry, and this is enough

Check the list of local images

**docker image ls**

And then the containers

**docker container ls**

Open a browser tab and navigate to <http://localhost:8080>

The application is working

Remove the container

**docker container rm --force other**

## Clean up

We can return our system to a clean state by removing all containers and images

First, we can stop all running containers

**docker container stop $(docker container ls -q)**

Then, we can remove all stopped containers

**docker container prune**

And finally, we can remove all images

**docker image rm $(docker image ls -q)**