# Solution M3: Advanced Docker

This is one possible and fully automated solution of the tasks included in the homework

All steps that follow assume that we decided to base our solution on Ubuntu Server 20.04 box we created in M1

Please note, that this solution is intentionally far from being an optimal one

Here, the emphasis is put on readability and not on optimization or speed

You must adjust some values like IP addresses, image (or box) names, repository names, credentials, etc. to match your use case

## The Environment

Prepare the environment by creating a **Vagrantfile** with the following content

# -\*- mode: ruby -\*-

# vi: set ft=ruby :

Vagrant.configure("2") do |config|

  config.ssh.insert\_key = false

  config.vm.define "docker1" do |docker1|

    docker1.vm.box="ubuntu-server"

    docker1.vm.hostname = "docker1.dob.lab"

    docker1.vm.network "private\_network", ip: "192.168.99.100"

    docker1.vm.synced\_folder "vagrant/", "/vagrant"

    docker1.vm.provision "shell", path: "docker.sh"

    docker1.vm.provision "shell", path: "docker-master.sh"

  end

  config.vm.define "docker2" do |docker2|

    docker2.vm.box="ubuntu-server"

    docker2.vm.hostname = "docker2.dob.lab"

    docker2.vm.network "private\_network", ip: "192.168.99.101"

    docker2.vm.synced\_folder "vagrant/", "/vagrant"

    docker2.vm.provision "shell", path: "docker.sh"

    docker2.vm.provision "shell", path: "docker-worker.sh"

  end

  config.vm.define "docker3" do |docker3|

    docker3.vm.box="ubuntu-server"

    docker3.vm.hostname = "docker3.dob.lab"

    docker3.vm.network "private\_network", ip: "192.168.99.102"

    docker3.vm.synced\_folder "vagrant/", "/vagrant"

    docker3.vm.provision "shell", path: "docker.sh"

    docker3.vm.provision "shell", path: "docker-worker.sh"

    docker3.vm.provision "shell", path: "docker-build-and-run.sh"

  end

end

We will need to create a **docker.sh** file to install and configure the necessary packages

#!/bin/bash

echo "\* Disable auto-update timers and service if present ..."

systemctl disable --now apt-daily-upgrade.timer &> /dev/null || true

systemctl disable --now apt-daily.timer &> /dev/null || true

systemctl disable --now unattended-upgrades.service &> /dev/null || true

sed -i s/1/0/g /etc/apt/apt.conf.d/20auto-upgrades || true

echo "\* Update repositories information ..."

apt-get update -y

echo "\* Install additional software ..."

apt-get install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common

echo "\* Install the Docker repository key ..."

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

echo "\* Add the Docker repository ..."

add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

echo "\* Install the Docker software ..."

apt-get update -y

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

echo "\* Disable and stop the Docker daemon ..."

systemctl disable --now docker

echo "\* Adjust Docker configuration ..."

cp /vagrant/daemon.json /etc/docker/daemon.json

mkdir -p /etc/systemd/system/docker.service.d/

cp /vagrant/docker.conf /etc/systemd/system/docker.service.d/docker.conf

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

systemctl enable --now docker

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

usermod -aG docker vagrant

Two more files can be seen here - **daemon.json** and **docker.conf**

The **daemon.json** file is used to change the default **Docker** configuration and enable the daemon to listen on all network interfaces for client (**Docker CLI**)connections

This way, we will be able to control it remotely

The **daemon.json** file should have the following content

{

    "hosts": ["tcp://0.0.0.0:2375", "unix:///var/run/docker.sock"]

}

The other file (**docker.conf**) is used to alter the way **Docker** daemon initializes

[Service]

ExecStart=

ExecStart=/usr/bin/dockerd

Next, we can create two separate files for the Swarm initialization and joining processes

First, we will create one for the master named **docker-master.sh** with the following content

#!/bin/bash

echo "\* Initialize swarm ..."

docker swarm init --advertise-addr 192.168.99.100

echo "\* Save the worker join token to a file ..."

docker swarm join-token -q worker > /vagrant/token.txt

Then, we will create a **docker-worker.sh** file for the nodes with the following content

#!/bin/bash

echo "\* Join the swarm ..."

docker swarm join --token $(cat /vagrant/token.txt) --advertise-addr $(hostname -I | cut -d ' ' -f 2) 192.168.99.100:2377

So far, we have the base infrastructure

We can stop here and do the steps related to the images and containers manually

Instead, we will continue our automation initiative

## The Web Component

In general, we should stick to the rule - one container, one process

This will lead us to a separation of the PHP and NGINX functionalities

Because the task requirements are asking us to combine PHP and NGINX in one container, let us see how this can be done

There are multiple ways to accomplish this

For this particular example the Supervisor utility was used

Once again, please note, that even though it is recommended every container to be responsible for one process (to have one process running), we can override this recommendation

We must create a **Dockerfile** with the following or similar content

FROM php:fpm

# Install additional packages

RUN docker-php-ext-install pdo\_mysql

# Adds the NGINX and Supervisor packages

RUN apt-get update -y && apt-get install -y nginx supervisor

# Removes the /etc/nginx/sites-{available,enabled} folders

RUN rm -rf /etc/nginx/sites-\*

# Redirects the NGINX log (access and error) to stdout and stderr

RUN ln -sf /dev/stdout /var/log/nginx/access.log && ln -sf /dev/stderr /var/log/nginx/error.log

# Adds our specific configuration for NGINX

ADD nginx-php.conf /etc/nginx/conf.d/nginx-php.conf

# Adds configuration for Supervisor utility

ADD supervisor.conf /etc/supervisor.conf

# Copies our PHP application

ADD php/index.php /var/www/html/index.php

# Exposes port 80

EXPOSE 80

# Sets the routine to be executed when container is started

CMD ["supervisord", "-c", "/etc/supervisor.conf"]

As we can see a few other files are referenced here

One of the additional files is the **NGINX** configuration (**nginx-php.conf**) with the following content

server {

        listen 80  default\_server;

        root /var/www/html;

        index index.php index.html index.htm;

        location / {

            try\_files $uri index.php$is\_args$args;

        }

        location ~ \.php$ {

            include fastcgi\_params;

            fastcgi\_pass 127.0.0.1:9000;

            fastcgi\_index index.php;

            fastcgi\_param SCRIPT\_FILENAME $document\_root/$fastcgi\_script\_name;

        }

}

The other additional file is the **Supervisor** configuration (**supervisor.conf**) with the following content

[supervisord]

nodaemon=true

[program:nginx]

command=nginx -c /etc/nginx/nginx.conf  -g 'daemon off;'

process\_name=%(program\_name)s\_%(process\_num)02d

numprocs=1

autostart=true

autorestart=false

startsecs=0

redirect\_stderr=true

stdout\_logfile=/dev/stdout

stdout\_logfile\_maxbytes=0

[program:php-fpm]

command=php-fpm -R -F -c /usr/local/etc/php-fpm.conf

process\_name=%(program\_name)s\_%(process\_num)02d

numprocs=1

autostart=true

autorestart=false

startsecs=0

redirect\_stderr=true

stdout\_logfile=/dev/stdout

stdout\_logfile\_maxbytes=0

The **index.php** file is the one used during the class practice, so nothing new here

## The Database Component

This component is simpler than the other one

Here we have just one process encapsulated in one container

The **Dockerfile** may look like this

FROM mysql:5.7

# Load the user data (our data) into the database

ADD ./sql/init.sql /docker-entrypoint-initdb.d/init.sql

The **init.sql** file is the one used during the class practice, so nothing new here

## Other Infrastructure Files

In order to achieve as much automation as possible, we will need a few more files

The main supplementary file is the **docker-build-and-run.sh** executed on the third node. It should be like

#!/bin/bash

echo "\* Login to Docker Hub ..."

cat /vagrant/docker-hub-cred.txt | docker login --username shekeriev --password-stdin

echo "\* Buld the web (php+nginx) image ..."

docker build -t shekeriev/img-dob-hw3-web /vagrant/web/.

echo "\* Build the db image ..."

docker build -t shekeriev/img-dob-hw3-db /vagrant/db/.

echo "\* Push the web image ..."

docker image push shekeriev/img-dob-hw3-web

echo "\* Push the db image ..."

docker image push shekeriev/img-dob-hw3-db

echo "\* Start the stack ..."

docker -H 192.168.99.100 stack up -c /vagrant/docker-compose.yml hw3

echo "\* Show services in the stack ..."

docker -H 192.168.99.100 stack ps hw3

We should create a file (**docker-hub-cred.txt**) that contains our credentials (password) for **Docker Hub**

YOUR DOCKER HUB PASSWORD

Substitute the string with your password

The rest of the instructions in the above file are building the two images and pushing them to Docker **Hub**

Then they can be referred in our **docker-compose.yml** file and used to spin up our application stack

## The Compose File

Finally, we reached the point to check the **docker-compose.yml** file

It should have the following content

version: "3"

services:

    swrm-web:

        image: shekeriev/img-dob-hw3-web

        deploy:

            replicas: 3

        ports:

            - 8080:80

        networks:

            - swrm-network

        depends\_on:

            - swrm-db

    swrm-db:

        image: shekeriev/img-dob-hw3-db

        networks:

            - swrm-network

        environment:

            MYSQL\_ROOT\_PASSWORD: "12345"

networks:

    swrm-network:

## Automatic Execution

This is all we need

The solution can be brought to life with the usual command

**vagrant up**

Sit back, relax, and watch the process

Once, everything is up and running, give it additional 30 seconds

Then open a browser tab on the host and navigate to one of the IP address of one of the nodes, for example

<http://192.168.99.100:8080>

There you should see the web application used for the class practice