**DevOps and Cloud**

**March 2025**



# ContainerOrchestration

Homework (M3)

Vasil Atanasov

@VasAtanasov

## Summary

## Docker Swarm Automation

This Vagrantfile defines and provisions a local Docker Swarm cluster using VirtualBox VMs, with a flexible number of manager and worker nodes. It automates the full setup of a Swarm environment, including token sharing, node joining, service deployment, and tool installation.

### Configuration & Constants

The SETTINGS map is used to set common values for all machines.

SETTINGS = {

BOX\_NAME: 'shekeriev/debian-12.11',

MEMORY: 2048,

CPUS: 1,

SUBNET: '192.168.99.0/24',

MANAGERS: 1,

WORKERS: 2,

SHARED\_DIR: '/vagrant'

}

There are other variables for configuration.

* DEBUG: Enables debug logging if DEBUG=true is set in the environment.
* IP\_START = 101: Defines the IPs start range

### Helper Methods

* ips(subnet, start, count): Generates a list of static IPs within a subnet.
* host\_entries(role, count, offset, ips): Generates /etc/hosts entries for all nodes.

### Common Provisioning Steps

First some common configurations and provisioning will be set up for all machines:

#### Common VirtualBox (RAM, CPU, group)

config.vm.provider :virtualbox do |vb|

vb.customize ['modifyvm', :id, '--memory', SETTINGS[:MEMORY]]

vb.customize ['modifyvm', :id, '--cpus', SETTINGS[:CPUS]]

vb.customize ['modifyvm', :id, '--groups', '/swarm']

end

#### A welcome logo

Notification on login for the role of the current node (Manager/Leader/Worker).

The logo script is in assets/logo file.

A black background with white letters

AI-generated content may be incorrect.

#### boostra.sh

Script that installs some common tools and common preparations like entries in the /etc/hosts file for all machines.

#### docker-setup.sh

Script that will install Docker Engine and dependencies on all machines. This is an important part for initializing a Docker Swarm cluster.

### Manager Node Logic

**The first manager will always be the leader.**

#### Common manager steps

* Set hostname
* Assing private IP address from the dynamically generated ip\_list based on the machine index.
* Set a shared folder so that all machines have access to the generated tokens
* For manager nodes there is also a shared folder where a local docker registry will save the pushed docker images.

#### Only on Swarm Leader

##### Initializes Docker Swarm

manager.vm.provision 'Init docker swarm', type: :shell do |shell|  
 shell.inline = "docker swarm init --advertise-addr #{ip}"

end if i == leader\_index

##### Extracts join tokens

When the manager(leader) is initiated worker.token and manager.token file are saved in the shared folder.

manager.vm.provision 'Extract worker token', type: :shell do |shell|  
 puts "Saving worker token at #{SETTINGS[:SHARED\_DIR]}/worker.token" if DEBUG

shell.inline = "docker swarm join-token -q worker > #{SETTINGS[:SHARED\_DIR]}/worker.token"

end if i == leader\_index

manager.vm.provision 'Extract manager token', type: "shell" do |shell|  
 puts "Saving manager token at #{SETTINGS[:SHARED\_DIR]}/manager.token" if DEBUG

shell.inline = "docker swarm join-token -q manager > #{SETTINGS[:SHARED\_DIR]}/manager.token"

end if i == leader\_index

##### Deploys swarm helper services

###### Portainer Stack

This is for monitoring the Docker Swarm using the browser:

if ! docker stack ls | grep -q portainer; then

echo "\* Deploying Portainer Stack For Docker Swarm..."

docker stack deploy -c /vagrant/portainer-agent-stack.yml portainer

fi

To access the Portainer go to the browser and enter https//localhost:9433. Set an admin password and explore the cluster.

A screenshot of a computer

AI-generated content may be incorrect.

Screens screenshot of a computer

AI-generated content may be incorrect.

###### Local Docker Registry

if ! docker service ls | grep -q registry; then

echo "\* Starting Local Docker Registry Service..."

docker service create \  
 --detach=true \  
 --name registry \  
 --publish published=5000,target=5000 \  
 --constraint=node.role==manager \  
 --mode replicated \  
 --mount type=bind,src=/mnt/registry,dst=/var/lib/registry \  
 --replicas=1 \  
 registry:2

fi

It starts in replication mode with 1 replica and the shared folder for images is mounter. All images pushed to the registry will be available for all nodes on the cluster. If the leader fails the registry will be started on other manager accessing the shared registry folder so no images are lost.

###### Visualizer

<https://github.com/dockersamples/docker-swarm-visualizer>

A simple visualization tool for docker swarm.

if ! docker service ls | grep -q visualizer; then

echo "\* Starting Visualizer Service..."

docker service create \  
 --detach=true \  
 --name=visualizer \  
 --publish=8001:8080 \  
 --constraint=node.role==manager \  
 --mount=type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \  
 --mode replicated \  
 --replicas=1 \  
 dockersamples/visualizer

fi

A screenshot of a computer

AI-generated content may be incorrect.

##### Install Docker Tools

Those are some helpful tools but for docker swarm ***dry*** seems to be quite helpful.

Dry - <https://github.com/moncho/dry>

manager.vm.provision 'Install docker tools', type: :shell do |shell|  
 shell.path = 'docker-tools.sh'

shell.env = {  
 'INSTALL\_LAZYDOCKER' => 'false',  
 'INSTALL\_DIVE' => 'false',  
 'INSTALL\_DRY' => 'true',  
 'INSTALL\_TRIVY' => 'false',  
 'INSTALL\_HADOLINT' => 'false',  
 'INSTALL\_PUSHRM' => 'false',  
 }  
end

##### Deploy BG App

Useful tmux commands - <https://tmuxcheatsheet.com/>

A tmux session starts with a script (deploy-stack.sh) that waits for all nodes to be available and active then builds and deploys the application (details for the application in the second part of the document).

Tmux is useful in this case because the session will be kept running in the background.

Because tmux starts with root user when logged in we need to switch to root user.

sudo -i

Then to see the logs we need to attach to the tmux session

tmux attach

A screenshot of a computer

AI-generated content may be incorrect.

A black and green background with white text

AI-generated content may be incorrect.

Then to see the application go to <http://localhost:8081>. If you refresh enough times you will be able to different id.

A green line with black text

AI-generated content may be incorrect.

##### Cleanup Trigger

The trigger is executed when machines are destroyed with vagrant destroy and deletes tokens and registry data.

manager.trigger.before :destroy do |trigger|  
 trigger.name = "Clean host files"

trigger.run = {  
 inline: <<~POWERSHELL,  
 @("manager.token", "worker.token", "registry\_data") | ForEach-Object {  
 if (Test-Path $\_) { Remove-Item $\_ -Recurse -Force }  
 }  
 POWERSHELL

}  
end if i == leader\_index

#### For Additional Managers

Join the swarm using manger.token.

### Worker Node Logic

There is not much logic for the worker nodes. The important thing is that they need to be joined to the cluster with the worker.token.

## Docker Compose

The BgApp is located under the bgapp folder.

❯ tree /F

Folder PATH listing

Volume serial number is E68F-8E4F

C:.

│ .env

│ docker-compose.yaml

│

├───db

│ db\_setup.sql

│ Dockerfile

│

└───web

bulgaria-map.png

config.php

Dockerfile

index.php

services:  
  
 web:  
 deploy:  
 mode: replicated

replicas: 3

image: 127.0.0.1:5000/web

build:  
 context: web

dockerfile: Dockerfile

ports:  
 - "8081:80"

volumes:  
 - "${PROJECT\_ROOT}:/var/www/html:ro"

networks:  
 - app-network

depends\_on:  
 - db  
  
 db:  
 image: 127.0.0.1:5000/db

build:  
 context: db

dockerfile: Dockerfile

networks:  
 - app-network

environment:  
 MYSQL\_ROOT\_PASSWORD\_FILE: /run/secrets/db\_root\_password

secrets:  
 - db\_root\_password  
  
secrets:  
 db\_root\_password:  
 external: true  
  
networks:  
 app-network:

### Key Notes

#### web service

##### Replication settings

deploy:  
 mode: replicated

replicas: 3

##### Image

Note that the repository is 127.0.0.1:5000. This is our local repository which is started in the manager.

image: 127.0.0.1:5000/web

If I login to worker1 for example I will be able to see that the images were pulled from the

local repository.

A screenshot of a computer

AI-generated content may be incorrect.

#### db service

##### secrets

A docker secret is created with shell provisioning and used for the db password.

DB\_SECRET=$(openssl rand -hex 16)  
echo "$DB\_SECRET" | docker secret create db\_root\_password -

secrets:  
 - db\_root\_password

#### Variable interpolation

The command docker stack deploy does not support variable interpolation. That is we need to interpolate the docker-compose.yaml file with the environment variables before suppling it to the docker stack deploy command. If we explore the deploy-stack.sh we will see:

docker compose "${compose\_args[@]}" config \  
 | sed -E '/published:/s/"//g;/^name:/d' \  
 | yq 'del(.services[].depends\_on) | del(.services[].build)' -y \  
 | docker stack deploy -c - "$STACK\_NAME"

The following command interpolates the docker-compose.yaml file with env from .env file or the environment.

docker compose "${compose\_args[@]}" config

There are some problems when deploying the stack from an interpolated file. For example, the published port is displayed as string but the docker stack deploy command expects an integer. The name field is also not allowed. With sed we can correct this.

sed -E '/published:/s/"//g;/^name:/d'

There are some warnings that depends\_on and build fields are ignored so with the yq tool we can remove them. The -y flag means output to yaml.

yq 'del(.services[].depends\_on) | del(.services[].build)' -y