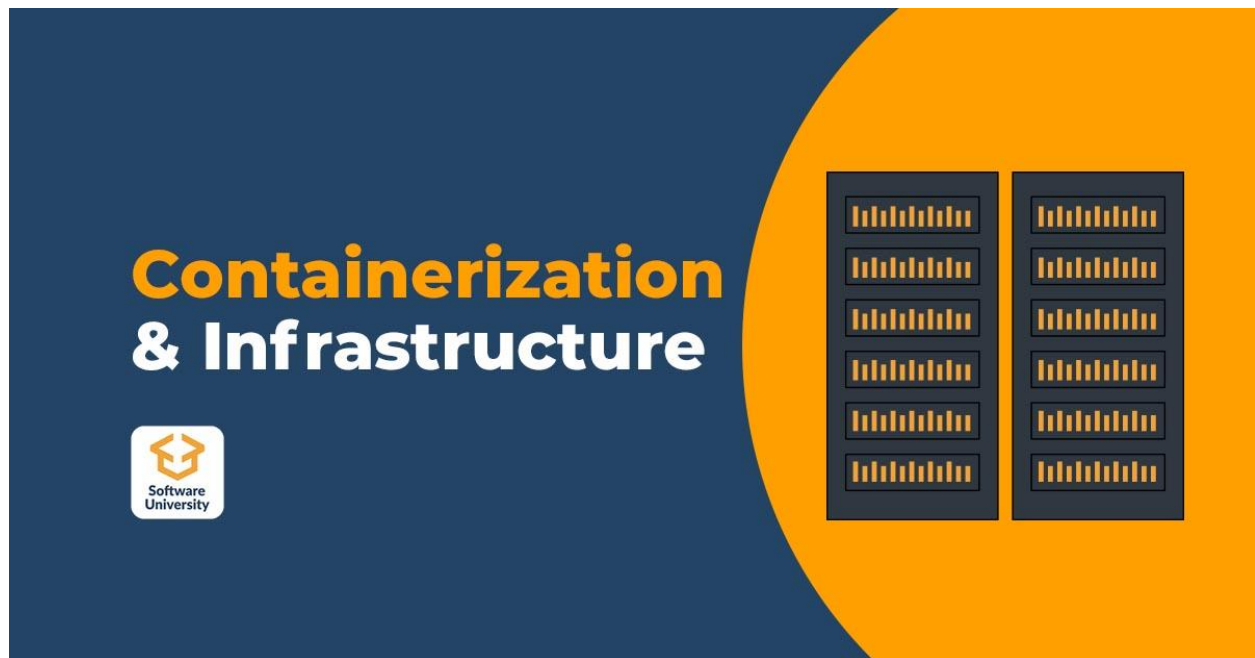


# DevOps and Cloud

March 2025



## Container Orchestration

Homework (M3)

Vasil Atanasov

@VasAtanasov

## 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.



### *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
- Assigning 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.



New Portainer installation

Please create the initial administrator user.

Username

admin

Password

Confirm password

X

The password must be at least 12 characters long.

Create user

☒ Allow collection of anonymous statistics. You can find more information about this in our [privacy policy](#).

Restore Portainer from backup

portainer.io  
COMMUNITY EDITION

Home

primary

Dashboard

Templates

Stacks

Services

Containers

Images

Networks

Volumes

Configs

Secrets

Swarm

Details

Setup

Registries

Administration

User-related

Environment-related

Registries

Logs

Notifications

Settings

portainer.io Community Edition 2.27.8 LTS

Display node labels

Refresh rate

5s

Cluster visualizer

manager

manager

CPU: 1

Memory: 2.06 GB

ready

lgapp\_web

Image: 127.0.0.1:5000/web:latest

Status: running

Update: 2025-06-30 11:11:09

portainer\_agent

Image: portainer/agent:th

Status: running

Update: 2025-06-30 11:28:06

portainer\_portainer

Image: portainer/portainer-ce:th

Status: running

Update: 2025-06-30 11:28:06

registry

Image: registry:2

Status: running

Update: 2025-06-30 11:05:25

visualizer

Image: dockersamples/visualizer:latest

Status: running

Update: 2025-06-30 11:06:18

worker1

worker

CPU: 1

Memory: 2.06 GB

ready

lgapp\_db

Image: 127.0.0.1:5000/db:latest

Status: running

Update: 2025-06-30 11:11:35

lgapp\_web

Image: 127.0.0.1:5000/web:latest

Status: running

Update: 2025-06-30 11:11:42

portainer\_agent

Image: portainer/agent:th

Status: running

Update: 2025-06-30 11:28:06

worker2

worker

CPU: 1

Memory: 2.06 GB

ready

lgapp\_web

Image: 127.0.0.1:5000/web:latest

Status: running

Update: 2025-06-30 11:11:30

portainer\_agent

Image: portainer/agent:th

Status: running

Update: 2025-06-30 11:28:06

5 | Page

## 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
```

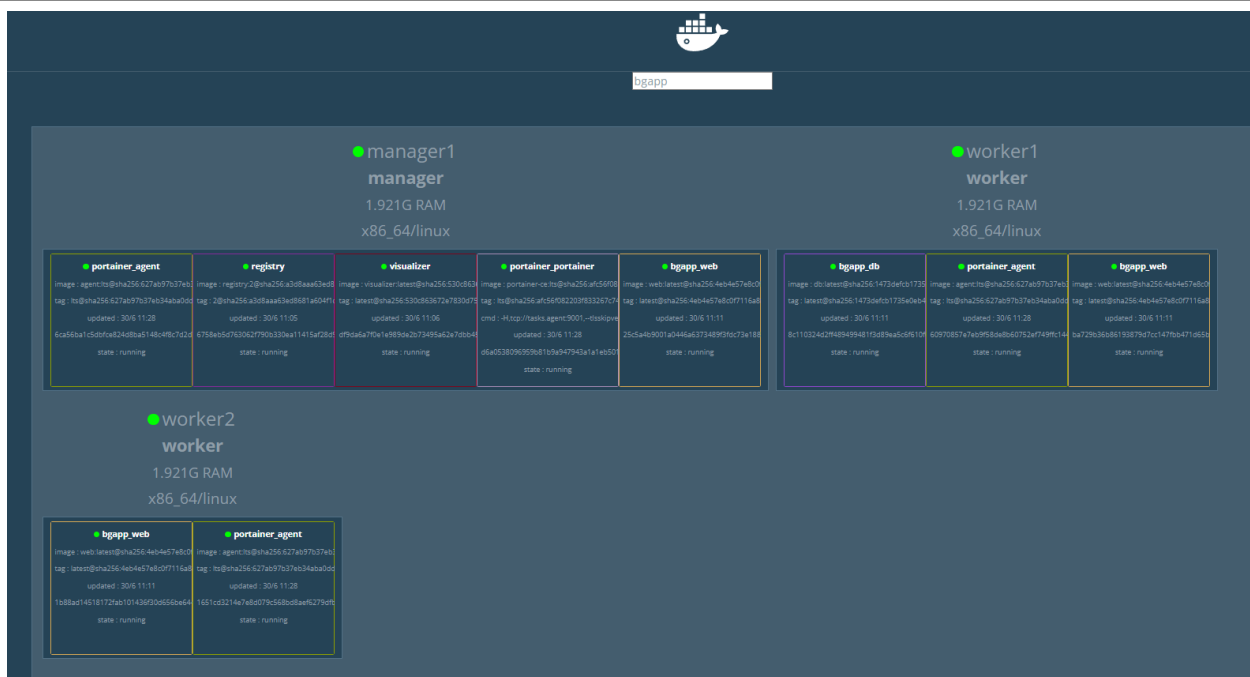
It starts in replication mode with 1 replica and the shared folder for images is mounter. All images pushed to the registry will be available for all nodes on the cluster. If the leader fails the registry will be started on other manager accessing the shared registry folder so no images are lost.

## Visualizer

<https://github.com/docker-samples/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
```



## 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
```

```
[INFO] Waiting for 3 nodes to be Ready...
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 1 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] 2 / 3 nodes Ready... retrying in 10
[INFO] All 3 nodes are Ready! Proceeding with stack deployment...
```

```

✓Pushing 127.0.0.1:5000/web: dc859a421064 Pushed
✓Pushing 127.0.0.1:5000/web: 7f072a7d1a8e Pushed
[*] Pulling 2/2
✓db Pulled
✓web Pulled
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network bgapp_app-network
Creating service bgapp_db
Creating service bgapp_web
root@manager1:/home/vagrant# vi /etc/ssh/sshd_config
[deploy_st0: tmux]*

```

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

## Факти за България

*Served by: ba729b36b861*



### 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:.\
  bgapp.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.

```
vagrant@worker1:~$ docker image ls
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
127.0.0.1:5000/web   <none>      e4ebdd79cfa8  3 minutes ago  514MB
127.0.0.1:5000/db    <none>      99814d0efc3e  3 minutes ago  326MB
portainer/agent      <none>      64b5f2e158ce  5 days ago    171MB
vagrant@worker1:~$ |
```

The Dockerfile creates an image from php:8.2-fpm and installs nginx as second service. To be able to start both php and nginx, supervisord is installed and used. This way Docker sees supervisord as the main process.

### *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 supplying 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
```