Оглавление

[1) Имена образов 1](#__RefHeading___Toc1822_4136223620)

[2) sha256.csv 1](#__RefHeading___Toc1824_4136223620)

[3) Расположение файлов на ftp 1](#__RefHeading___Toc1826_4136223620)

[4) NVIDIA 2](#__RefHeading___Toc1828_4136223620)

[5) Как сократить размер образам 3](#__RefHeading___Toc1830_4136223620)

#### 1) Имена образов

- в .env (APP\_IMAGE, APP\_TAG), в sha256.scv одинаковые <app\_image>:<tag>  
- имя архива такое же, двоеточие заменяется на подчёркивание <app\_image>\_<tag>.tar.gz  
- образы загруженные из архива (docker load -i <app\_image>\_<tag>.tar.gz), должны иметь имя ${REGISTRY\_Url}/${VENDOR}/${APP\_IMAGE}:${APP\_TAG}. Это имя складывается из переменных в .env и используется в compose

#### 2) sha256.csv

- docker inspect --format='{{index .Id}}' <app\_image>:<tag>

#### 3) Расположение файлов на ftp

vendor\_name

├── liveness\_engine\_cpu\_v3.6.2.tar.gz  
├── liveness\_engine\_cpu\_v3.6.3.tar.gz  
├── verify\_engine\_cpu\_v1.tar.gz

├── verify\_engine\_cpu\_v3.6.2.tar.gz  
├── verify\_proxy\_v0.1.tar.gz

├── liveness\_proxy\_v0.1.tar.gz

├── v1

│ ├── verification-cpu

│ │ ├── docker-compose.yml

│ │ ├── .env

│ │ ├── sha256  
│ │ ├── README  
│ │ ├── licence

├── v3.6.2

│ ├── liveness-voice-cpu

│ │ ├── docker-compose.yml

│ │ ├── .env

│ │ ├── sha256

│ ├── verification-cpu

│ │ ├── docker-compose.yml

│ │ ├── .env

│ │ ├── sha256

├── v3.6.3

│ ├── liveness-voice-cpu

│ │ ├── docker-compose.yml

│ │ ├── .env

│ │ ├── sha256  
В таком виде для того чтоб отработал наш скрипт автоматизировано развёртывания, тестирования. Логика скрипта примерно такая:

- запускаем скрипт с указание каталога с вашим compose файлом на ftp

- из этого каталога берется sha256.csv и читаются имена образов

- из каталога 2 уровнями выше берутся tar.gz из sha256.csv, проверяются контрольные суммы и образы заливаются в наш registry

- правиться .env, изменяется REGISTRY\_URl на наш

- правиться .env, выставляется CORE\_COUNT=1

- запускается нагрузка с увеличением потоков нагрузки

- записывается время ответа при 1 реплике БП и 1 потоке нагрузки (avgmin)

- записывается максимальная утилизация CPU

- высчитывается сколько реплик БП надо чтоб утилизировать 24 ядра (maxrepl)

- правиться .env, выставляется CORE\_COUNT=maxrepl

- запускается нагрузка с увеличением потоков

- находим количество потоков нагрузки при котором время ответа БП > avgmin\*1.5

- ...

#### 4) NVIDIA

#### Установка

## Gentoo

eselect enable vowstar

eix-update

emerge -av nvidia-container-toolkit

# В одиночном режиме все заработает, но для swarm еще нужен файл nvidia-container-runtime, хз почему он не ставится из порта nvidia-container-toolkit

# Скачиваем nvidia-container-toolkit-1.9.0-1.x86\_64.rpm (версию берем как установили с порта) и закидываем в /usr/bin/nvidia-container-runtime

# https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#id3

## RedOS7.3 (Centos7)

# Репа

curl -s -L https://nvidia.github.io/libnvidia-container/centos7/libnvidia-container.repo > /etc/yum.repos.d/libnvidia-container.repo

# Ядро

dnf install redos-kernels-release

dnf update

# установка

yum install nvidia-container-toolkit nvidia-kmod nvidia-modprobe nvidia-persistenced xorg-x11-drv-nvidia-cuda xorg-x11-drv-nvidia-cuda-libs

# Загрузка

# Edit /etc/default/grub. Append the following to “GRUB\_CMDLINE\_LINUX” rd.driver.blacklist=nouveau nouveau.modeset=0

# Generate a new grub configuration to include the above changes.

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

# Edit/create /etc/modprobe.d/blacklist.conf and append: blacklist nouveau

#### Настройка

## /etc/docker/daemon.json

# в node-generic-resources прописываем ID своих видеокарт

# nvidia-smi -a | grep UUID | awk '{print "NVIDIA-GPU="substr($4,0,12)}'

# https://gist.github.com/tomlankhorst/33da3c4b9edbde5c83fc1244f010815c?permalink\_comment\_id=3641014#gistcomment-3641014

# https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/user-guide.html#daemon-configuration-file

{

"runtimes": {

"nvidia": {

"path": "/usr/bin/nvidia-container-runtime",

"runtimeArgs": []

}

},

"default-runtime": "nvidia",

"node-generic-resources": [

"NVIDIA-GPU=GPU-c0fb513c"

]

}

## /etc/nvidia-container-runtime/config.toml

# разкоментируем строку swarm-resource = "DOCKER\_RESOURCE\_GPU"

### Проверка

## Run

docker run --rm --gpus all,capabilities=utility nvidia/cuda:11.8.0-base-ubuntu22.04 nvidia-smi

## Service

# https://docs.docker.com/engine/reference/commandline/service\_create/#create-services-requesting-generic-resources

docker service create --generic-resource "NVIDIA-GPU=0" --replicas 1 --name nvidia-cuda --entrypoint "sleep 5000" nvidia/cuda:11.8.0-base-ubuntu22.04

docker exec -it $(docker service ps --no-trunc --format "{{.Name}}.{{.ID}}" nvidia-cuda) nvidia-smi

docker service rm nvidia-cuda

## Stack

echo 'version: "3.5"

services:

cuda:

image: nvidia/cuda:11.8.0-base-ubuntu22.04

command: "sleep 5000"' > cuda-stack.yml

docker stack deploy -c cuda-stack.yml nvidia

docker exec -it $(docker service ps --no-trunc --format "{{.Name}}.{{.ID}}" nvidia\_cuda) nvidia-smi

docker stack rm nvidia

#### 5) Как сократить размер образа

# посмотреть инфу о слоях в образе:

- docker history

- docker run --rm -it -v /var/run/docker.sock:/var/run/docker.sock wagoodman/dive:latest <имя\_образа>

# Dockerfile

- Группируйте ваши команды  
- Используете multi-stage builds

- Применяйте опцию —squash при docker build