



Същност на Виртуализацията

Контейнери. Docker

Създаване на Контейнери с Dockerfile

Docker Compose

Оркестрация



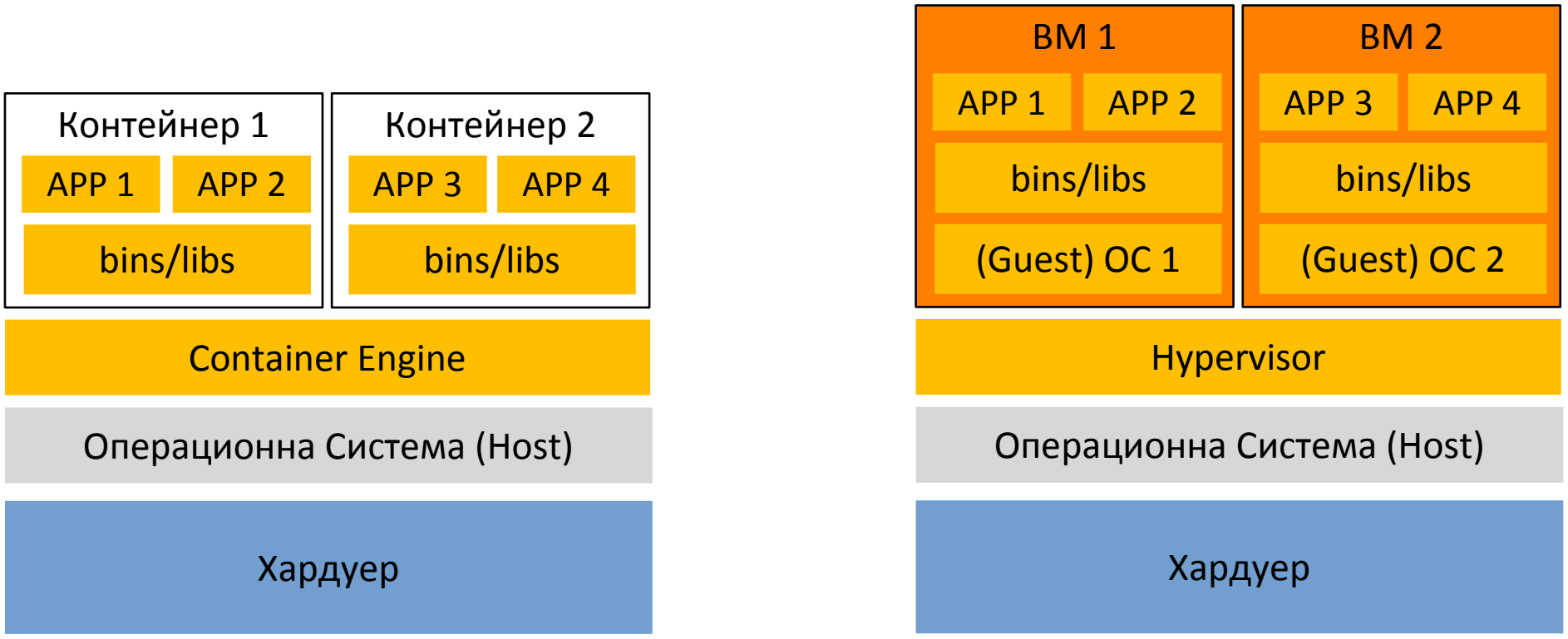
доц. д-р А. Пенев

Съдържание

1. Модел за работа със софтуерни Контейнери
2. Работа с Docker
3. Създаване на Контейнери с Dockerfile
4. Работа с Docker Compose
5. Оркестрация

Модел на Работа с Контейнери

Контейнери vs. Виртуални машини



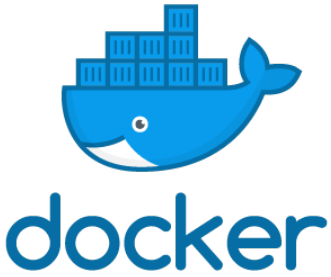
Контейнери

- **Контейнер** ще наричаме изолирана инстанция на потребителското пространство на ОС;
- Контейнерите са пример за OS-Level Виртуализация;
- Контейнерите обикновено съществуват благодарение на ядрото на ОС или друг софтуер (**Container engine**), който разпределя и изолира ресурсите на ОС в отделни пространства на имената (name-spaces);
- Контейнерите обикновено се изпълняват директно върху физическата машина, но приложенията в тях виждат и използват само част от ресурсите (процесор, памет, файлове и папки, В/И устройства и др.);

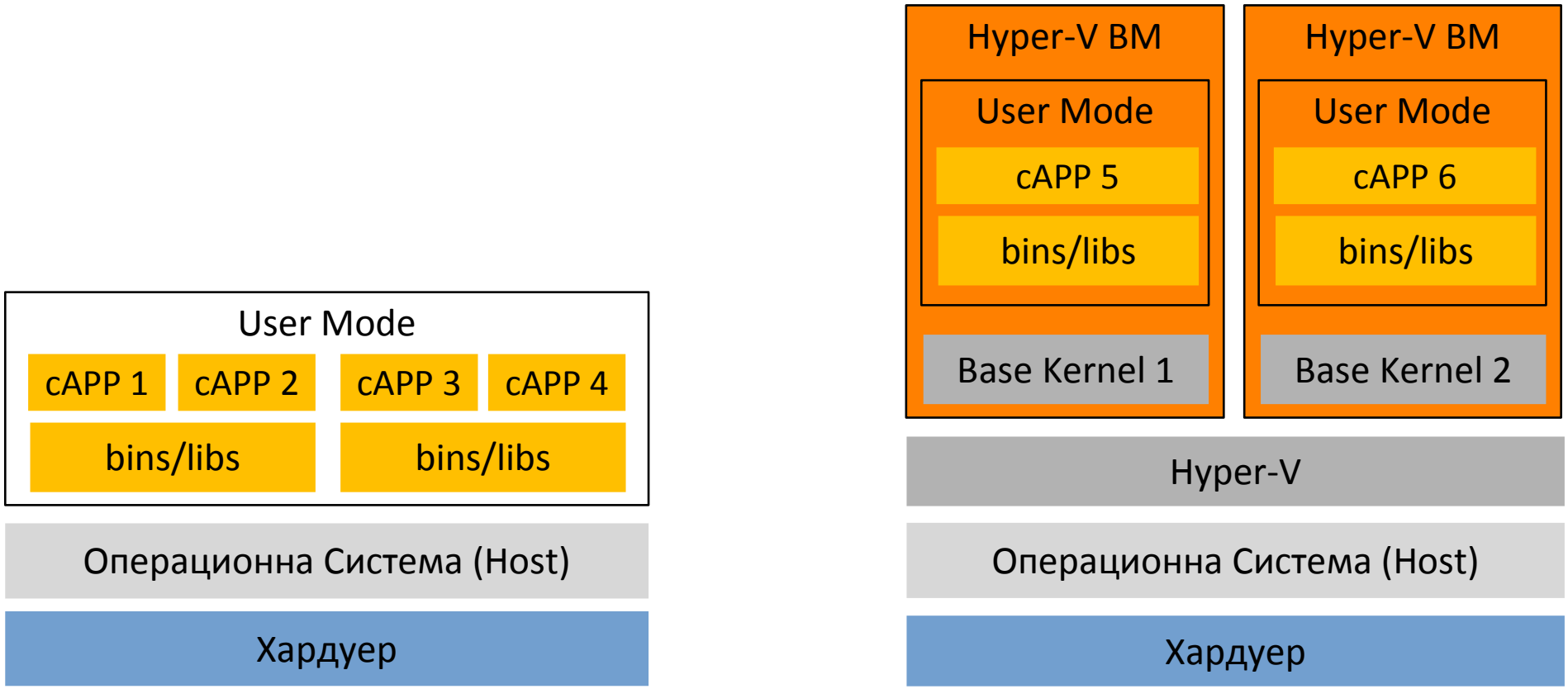
Контейнери (Container Engines) – реализации на концепцията



- Контейнери (Containers) – Solaris, Docker;
- Зони (Zones) – Solaris;
- Виртуални частни сървъри (Virtual private servers) – OpenVZ;
- Partitions, Virtual environments (VEs), Virtual kernel – DragonFly BSD;
- “Затвори” (Jails) – FreeBSD;



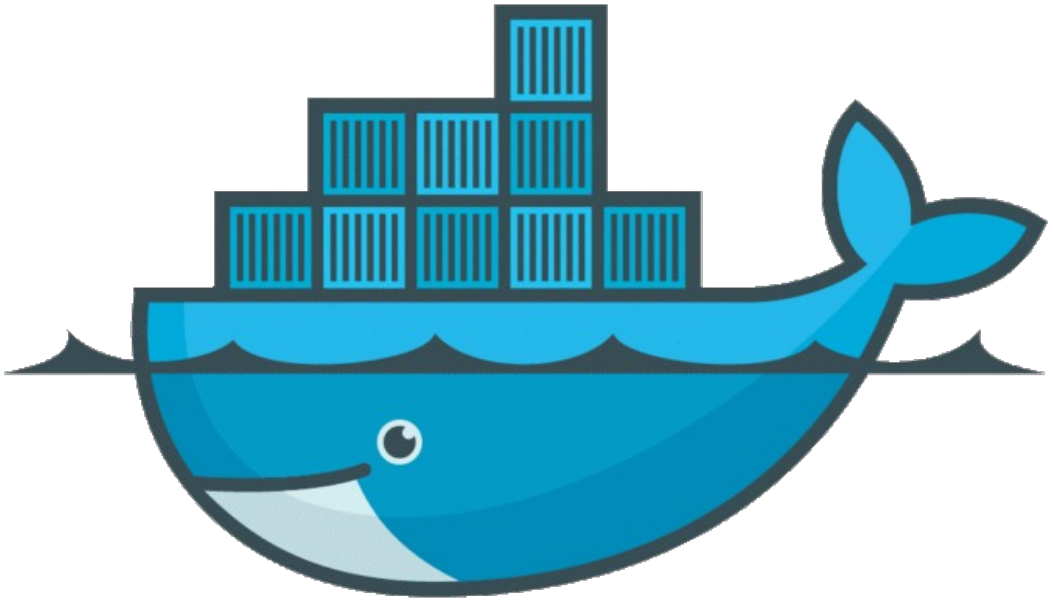
Windows Server Containers vs. Hyper-V Containers



Docker

Docker – еко система

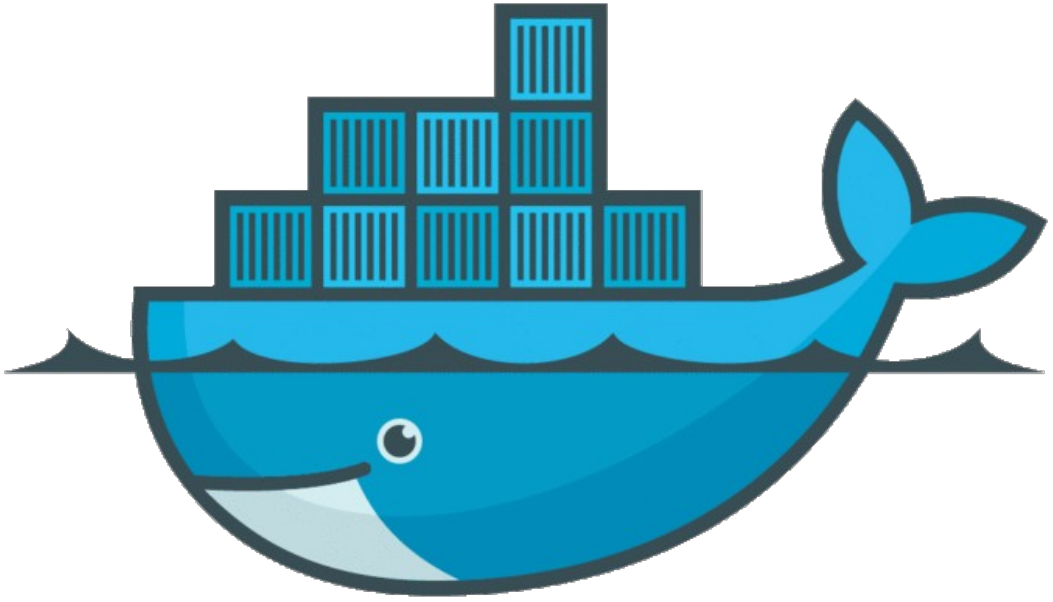
- **Docker Engine** – клиент-сървър приложение съставено от Docker Client и Docker Daemon;
- **Docker Networking** – позволява връзката между контейнерите, дори когато са на различни машини;
- **Docker Swarm** – управлява и оркестрира клъстер от Docker Engines;
- **Docker Desktop** – приложение, упростяващо използването на Docker на Windows и MacOS;



docker

Docker – еко система

- **Docker Compose** – инструмент за дефиниране и изпълнение на многоконтейнерни Docker приложения;
- **Docker Image** – съдържа всичко необходимо (код, библиотеки, обкръжение, конфигурация) за да се стартира приложение като контейнер;
- **Docker Hub** – онлайн регистър на Docker Images;



Docker – еко система

- **Docker Volume** – персистентно съхранение на данни;
- **Docker file** – Инструкции как да бъде създаден Docker Image;
- **Docker Container** – run-time инстанция на Image;

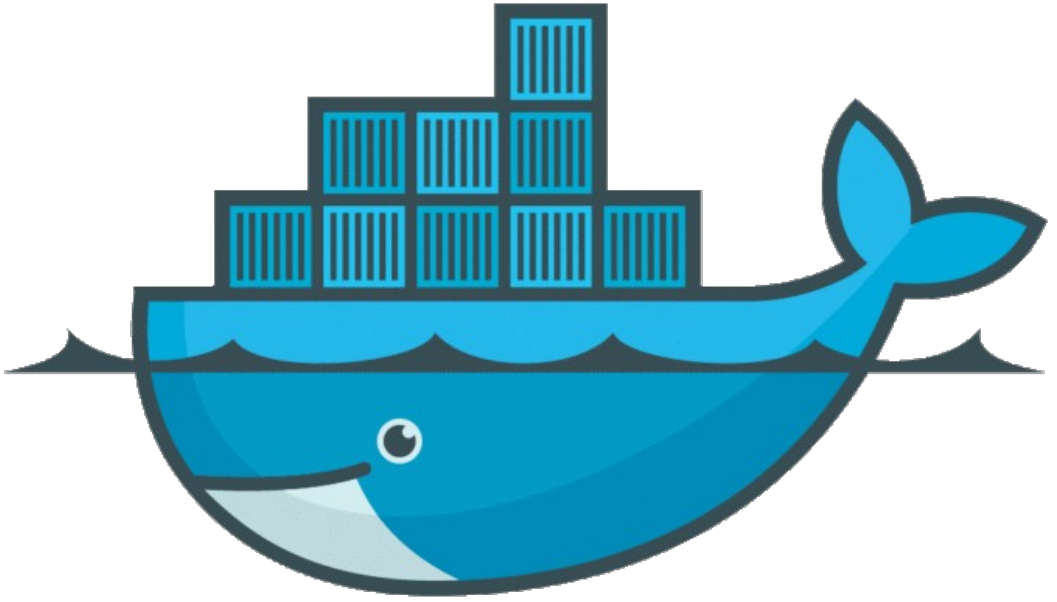
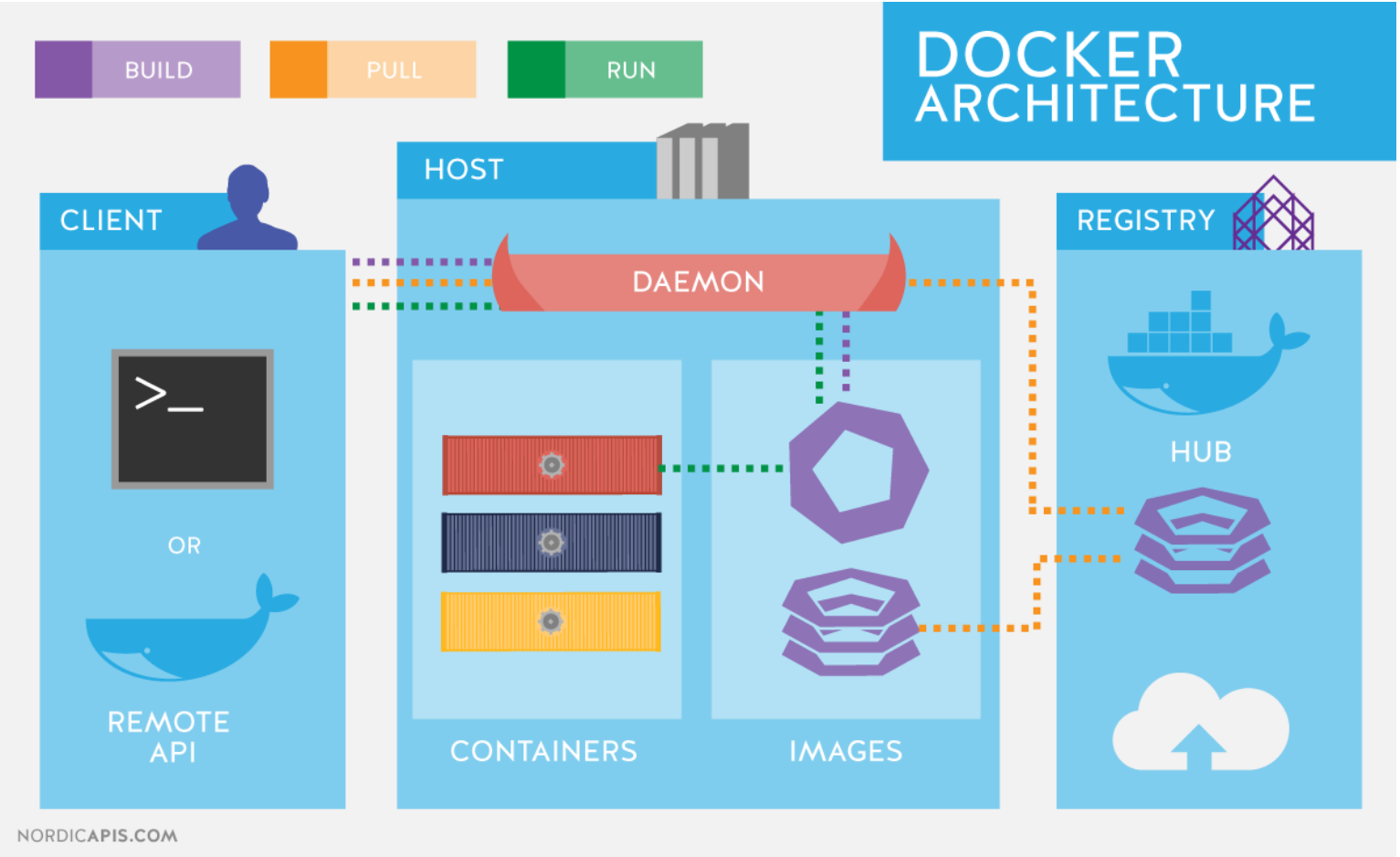
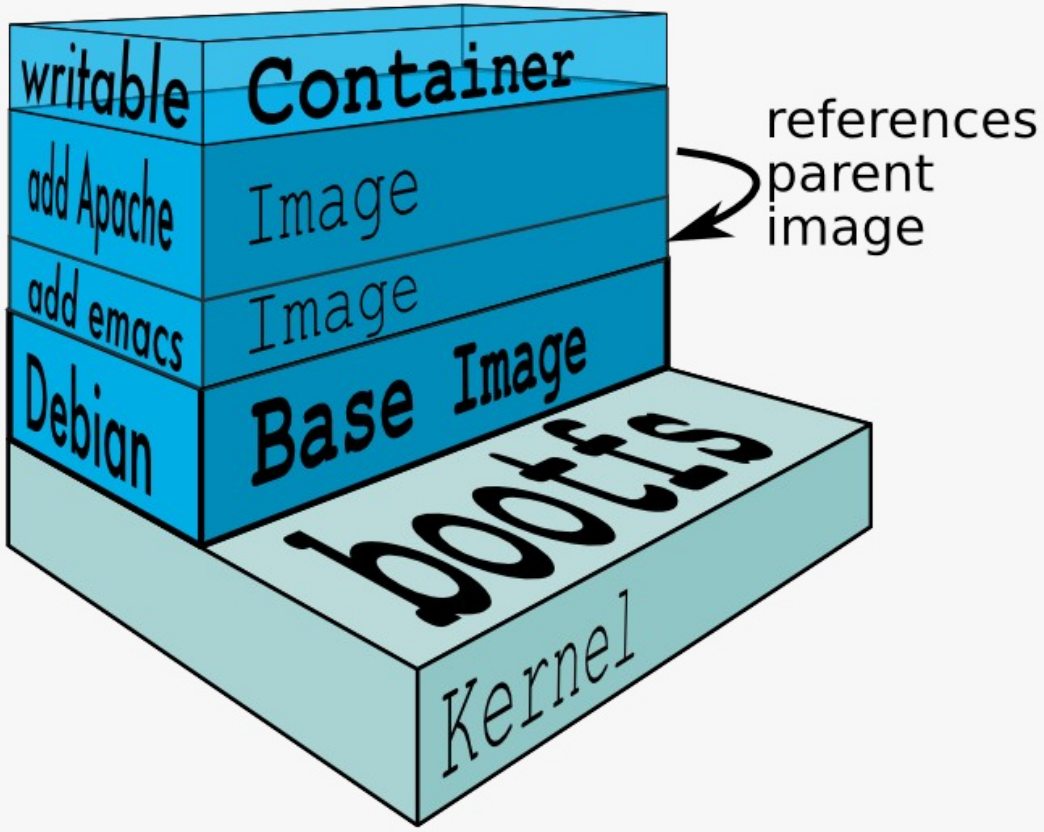


Схема на работа с Docker



Docker Image

- Read-only шаблон от който се създават контейнери;
- Ако вашето приложение се нуждае конкретни ОС, Apache сървър или Java това може да бъде събрано в един Docker Image;
- Docker Images са изградени от много (read-only) слоеве. Това позволява нови имиджи да се изграждат на базата на други;
- Когато създадем контейнер, за него се създава тънък презаписваем слой;



Docker – пример „Здравей, докер!“

```
$ docker --version
Docker version 19.03.1

$ docker run hello-world

docker : Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest:
sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

Docker – пример „Здравей, докер!“

```
$ docker image ls

$ docker container ls

$ docker container ls --all
CONTAINER ID  IMAGE        COMMAND    CREATED         STATUS                     PORTS          NAMES
f61e19d3143a  hello-world  "/hello"   34 seconds ago  Exited (0) 32 seconds ago        focused_hugle
```

Docker – пример 2 (Ubuntu)

```
$ docker run --interactive --tty ubuntu bash
docker : Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
22e816666fd6: Pull complete
079b6d2a1e53: Pull complete
11048ebae908: Pull complete
c58094023a2e: Pull complete
Digest: sha256:a7b8b7b33e44b123d7f997bd4d3d0a59fafc63e203d17efedf09ff3f6f516152
Status: Downloaded newer image for ubuntu:latest

root@8aea0acb7423:/# hostname
8aea0acb7423

root@8aea0acb7423:/# exit

$
```

Docker – пример 2 (Ubuntu 18.04)

```
$ docker pull ubuntu:18.04

$ docker images ls

$ docker run ubuntu:18.04

$ docker container ls --all
```

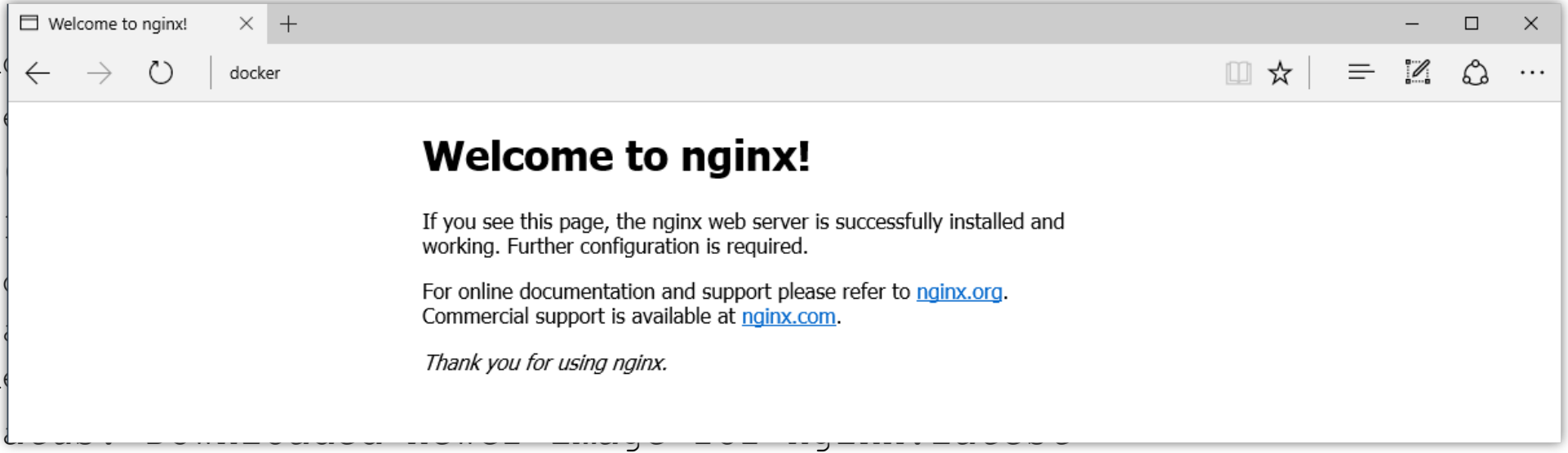
Docker – пример 3 (Nginx)

```
$ docker run --detach --publish 80:80 --name webserver nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx

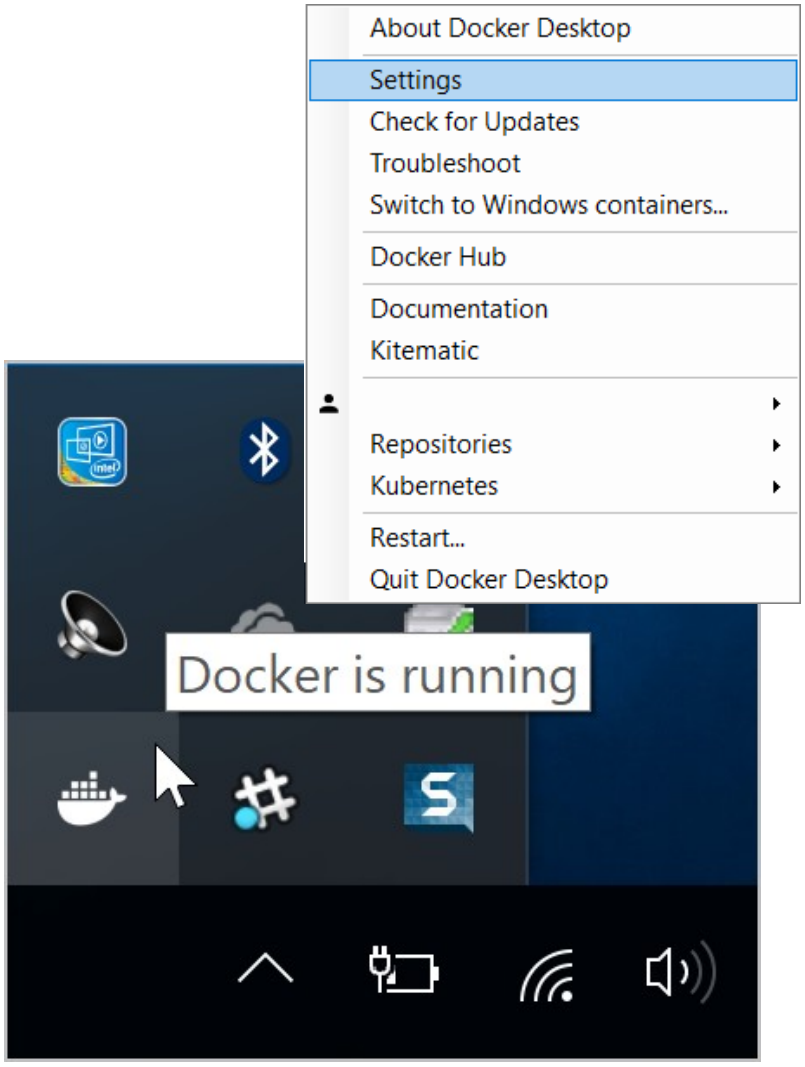
fdd5d7827f33: Pull complete
a3ed95caeb02: Pull complete
716f7a5f3082: Pull complete
7b10f03a0309: Pull complete
Digest:
sha256:f6a001272d5d324c4c9f3f183e1b69e9e0ff12debeb7a092730d638c33e
0de3e
Status: Downloaded newer image for nginx:latest
dfe13c68b3b86f01951af617df02be4897184cbf7a8b4d5caf1c3c5bd3fc267f
```

Docker – пример 3 (Nginx)

```
$ docker run --detach --publish 80:80 --name webserver nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fd
a3
71
7b
Di
sh
0d
St
dfe13c68b3b86f01951af617df02be4897184cbf7a8b4d5caf1c3c5bd3fc267f
```

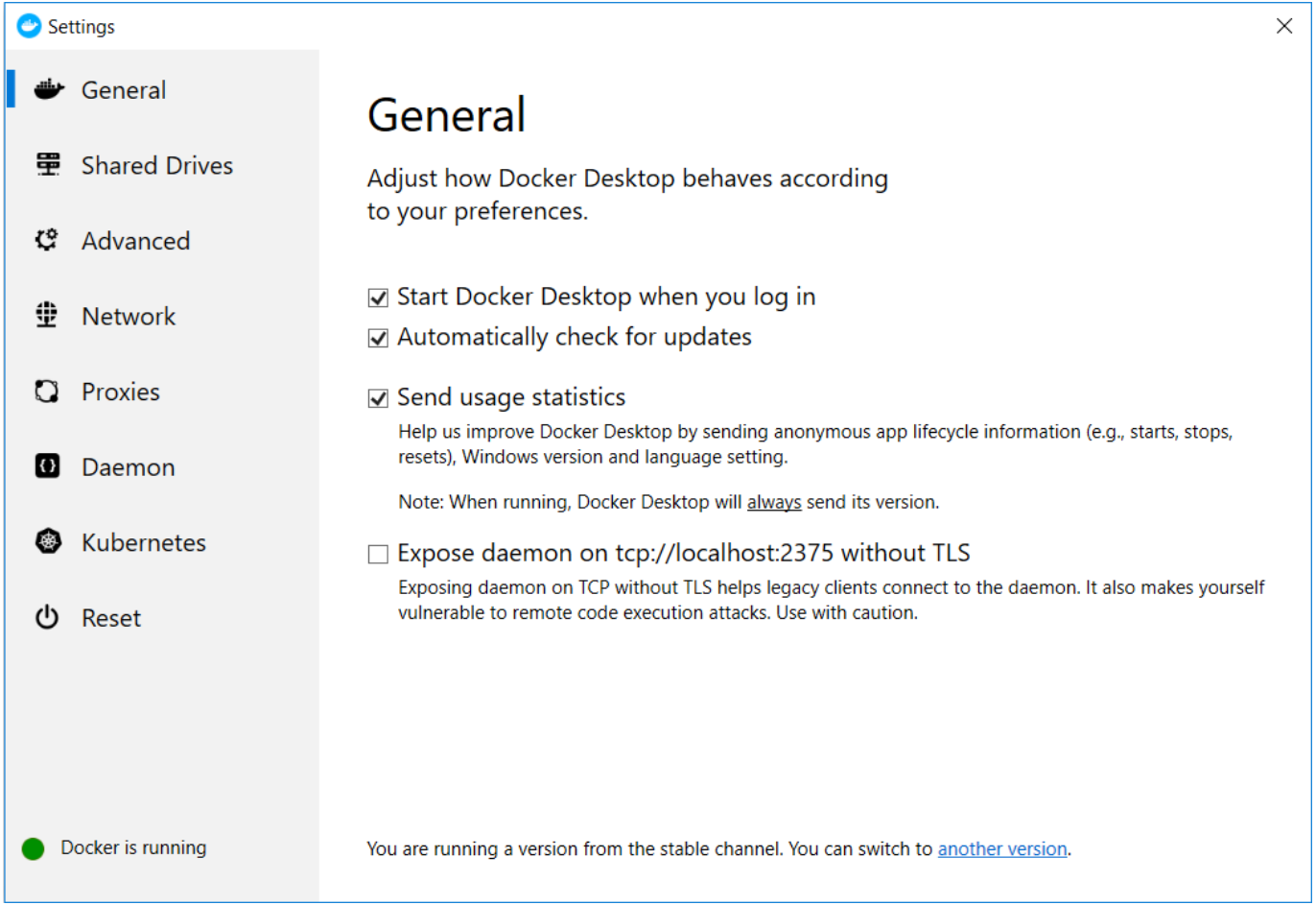


Docker Desktop – настройки



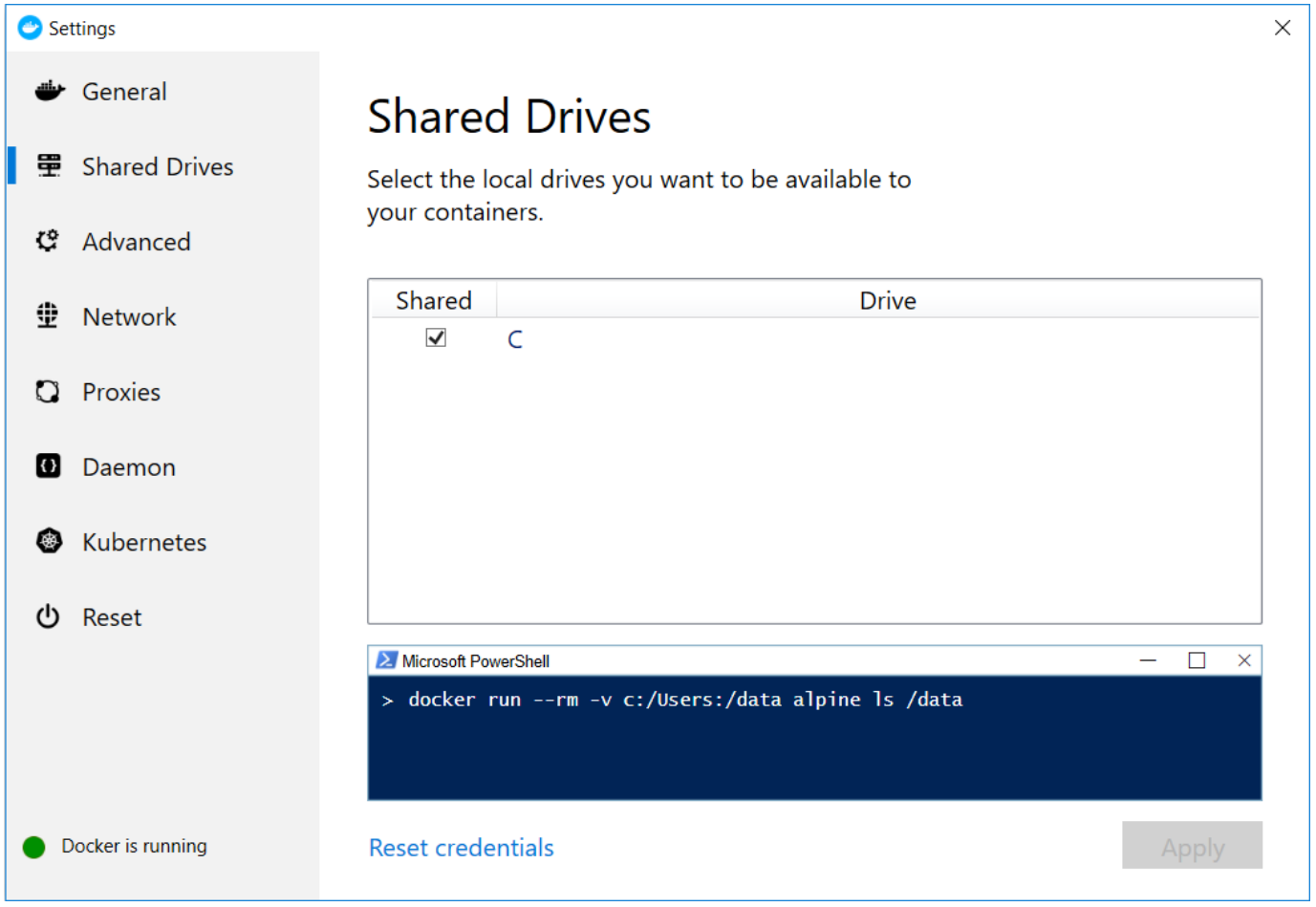
Избираме Settings

Docker Desktop – настройки



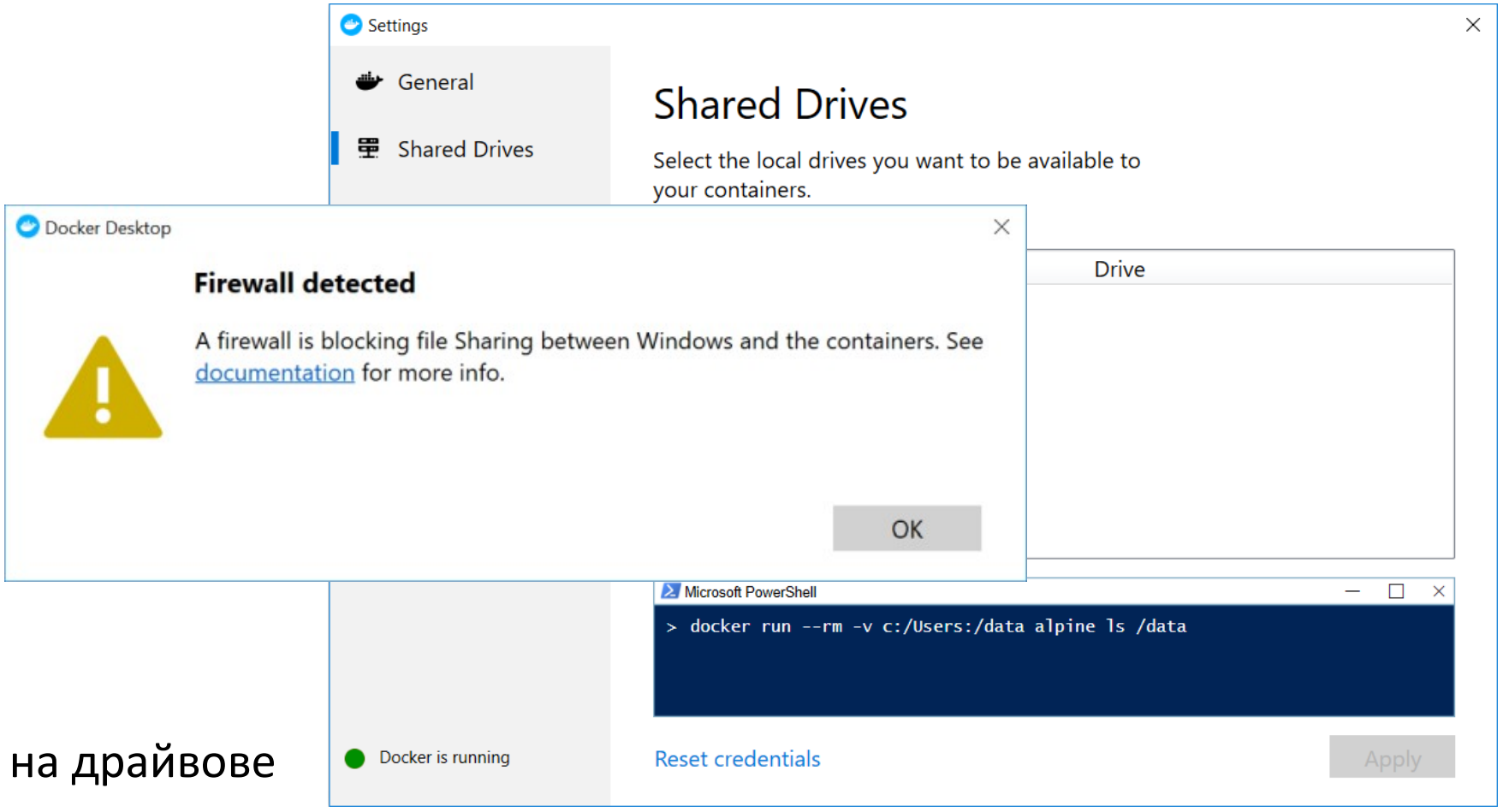
Основни настроки

Docker Desktop – настройки



Споделяне на драйвове

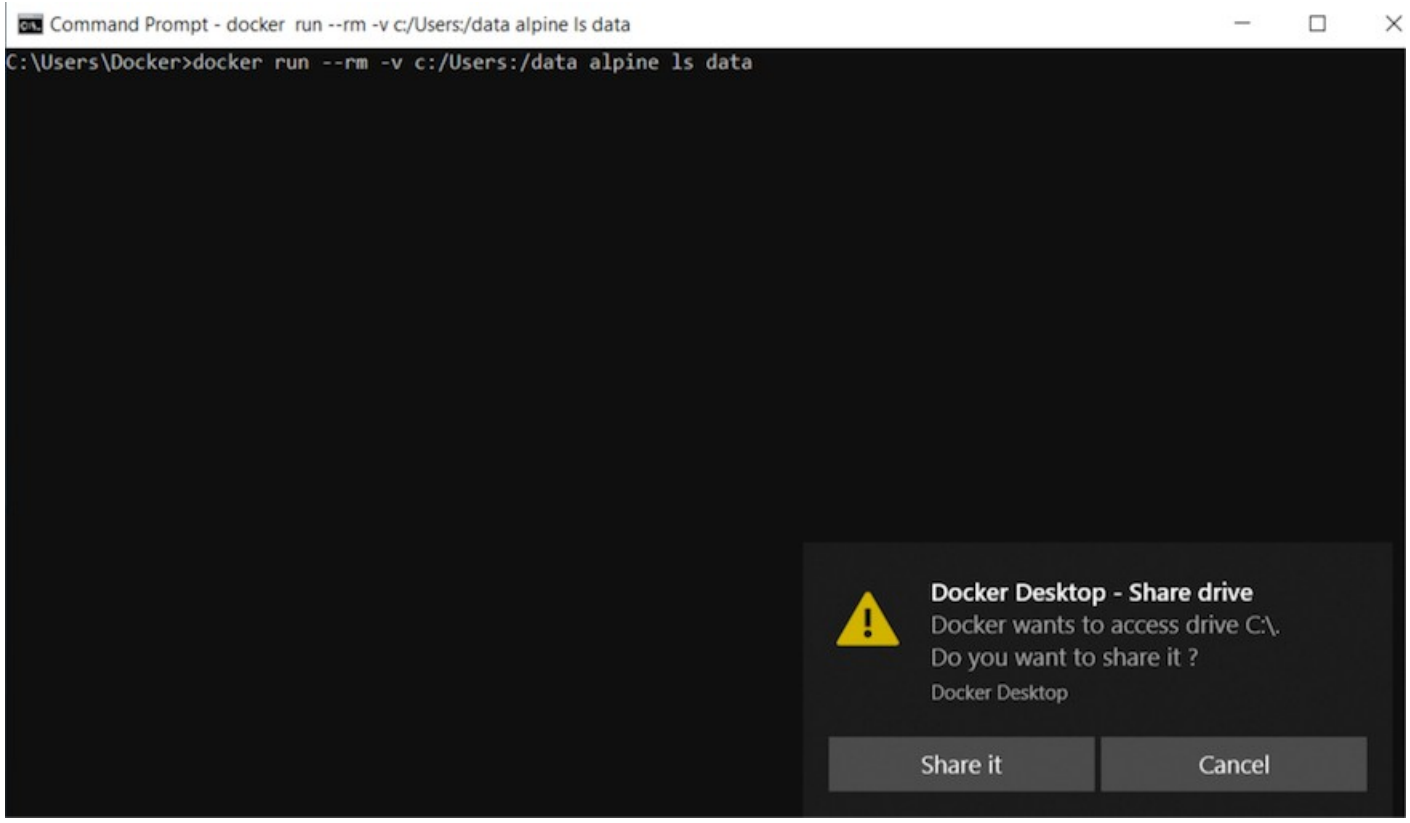
Docker Desktop – настройки



Споделяне на драйвове

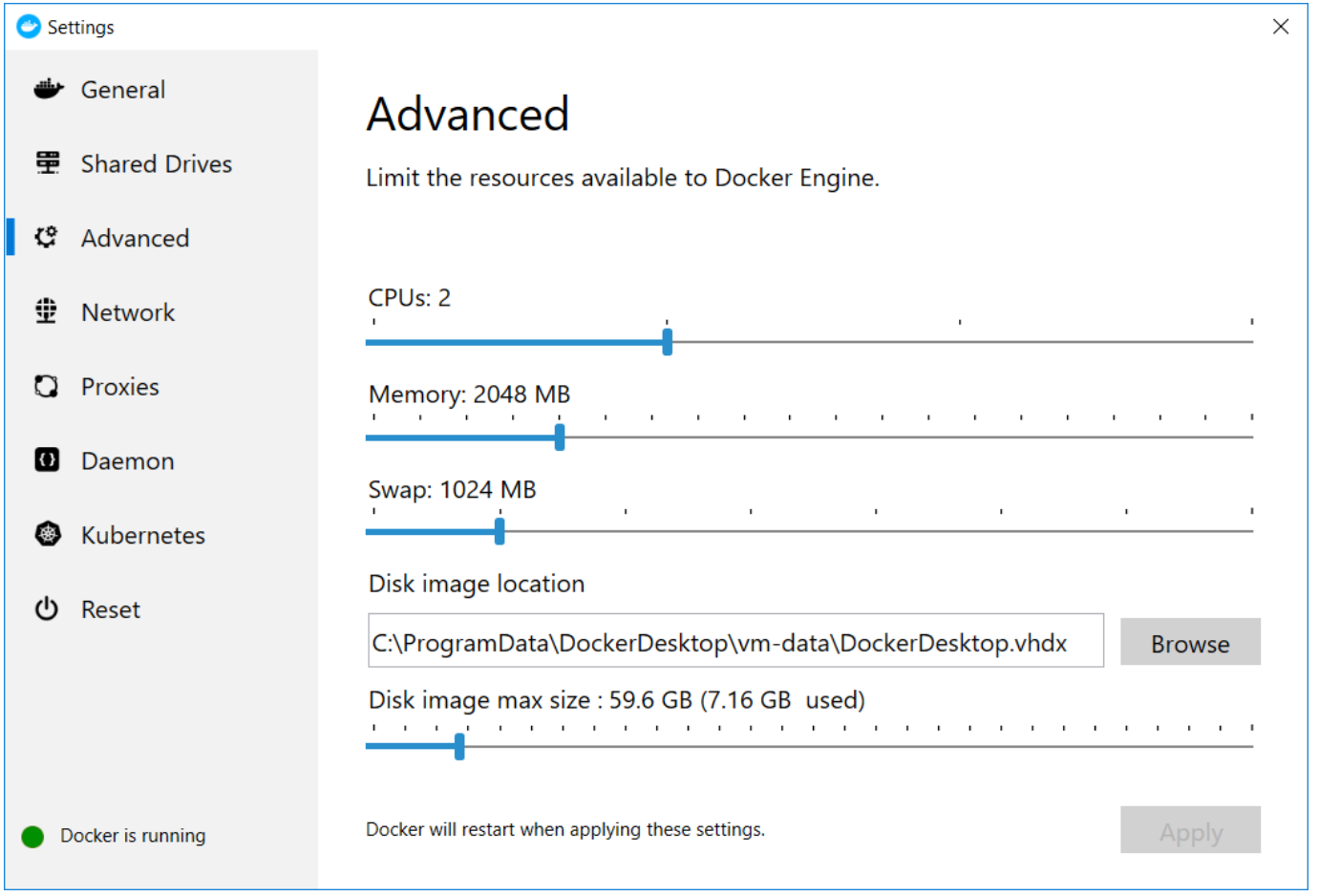
Docker Desktop – настройки

Потвърждаваме

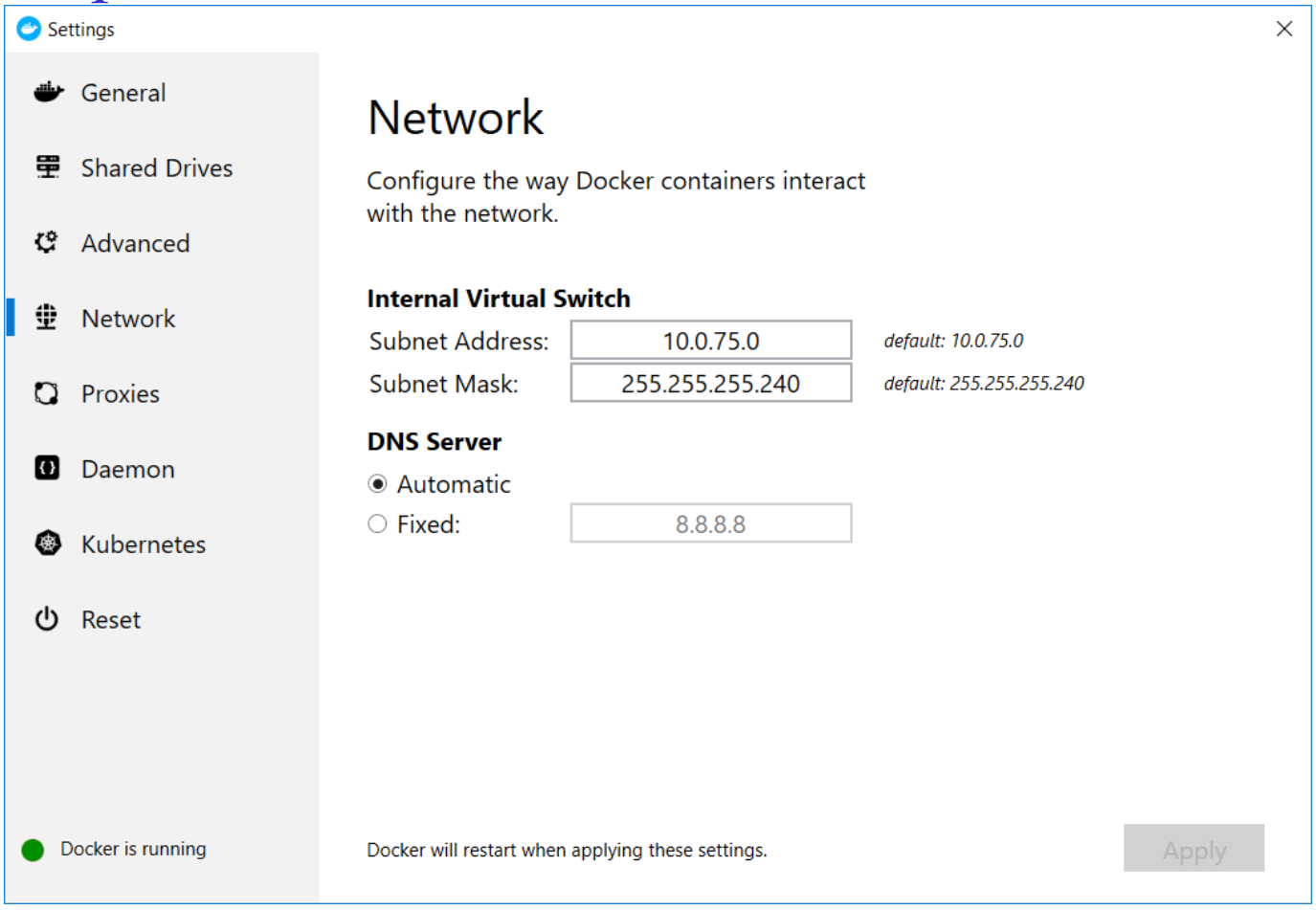


Docker Desktop – настройки

Финни настройки на процесор, памет, диск и др.

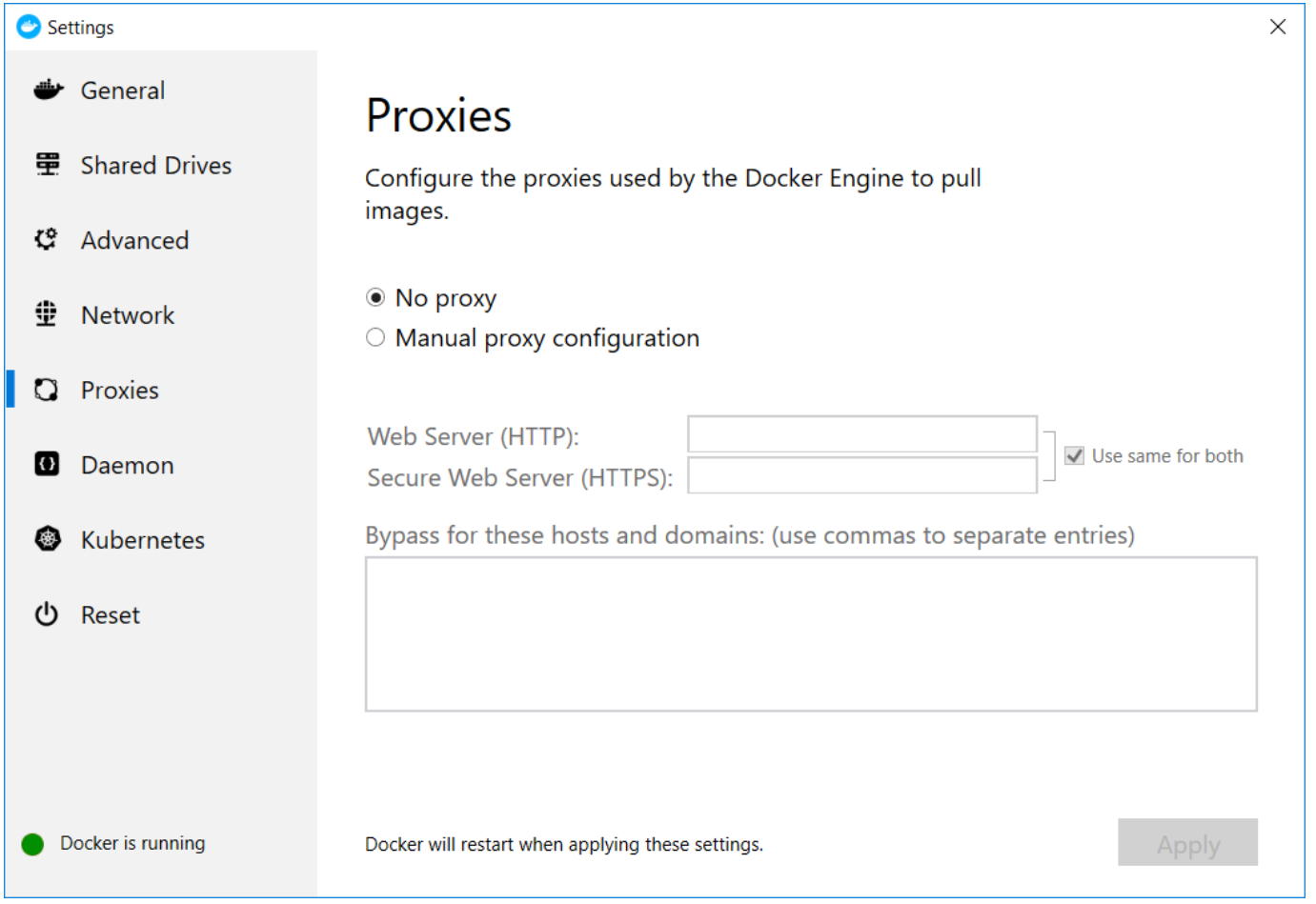


Docker Desktop – настройки



Настроки на мрежата

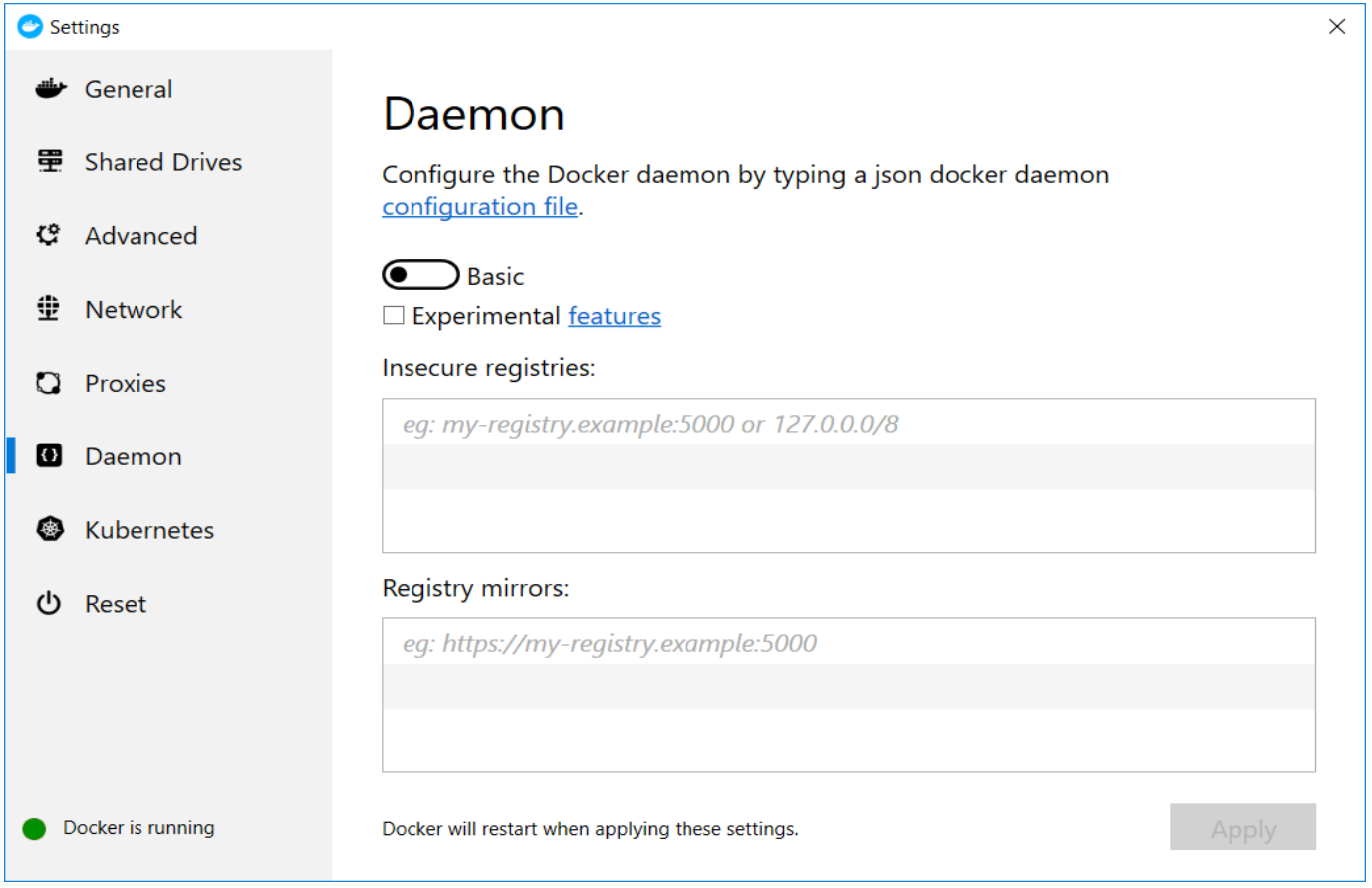
Docker Desktop – настройки



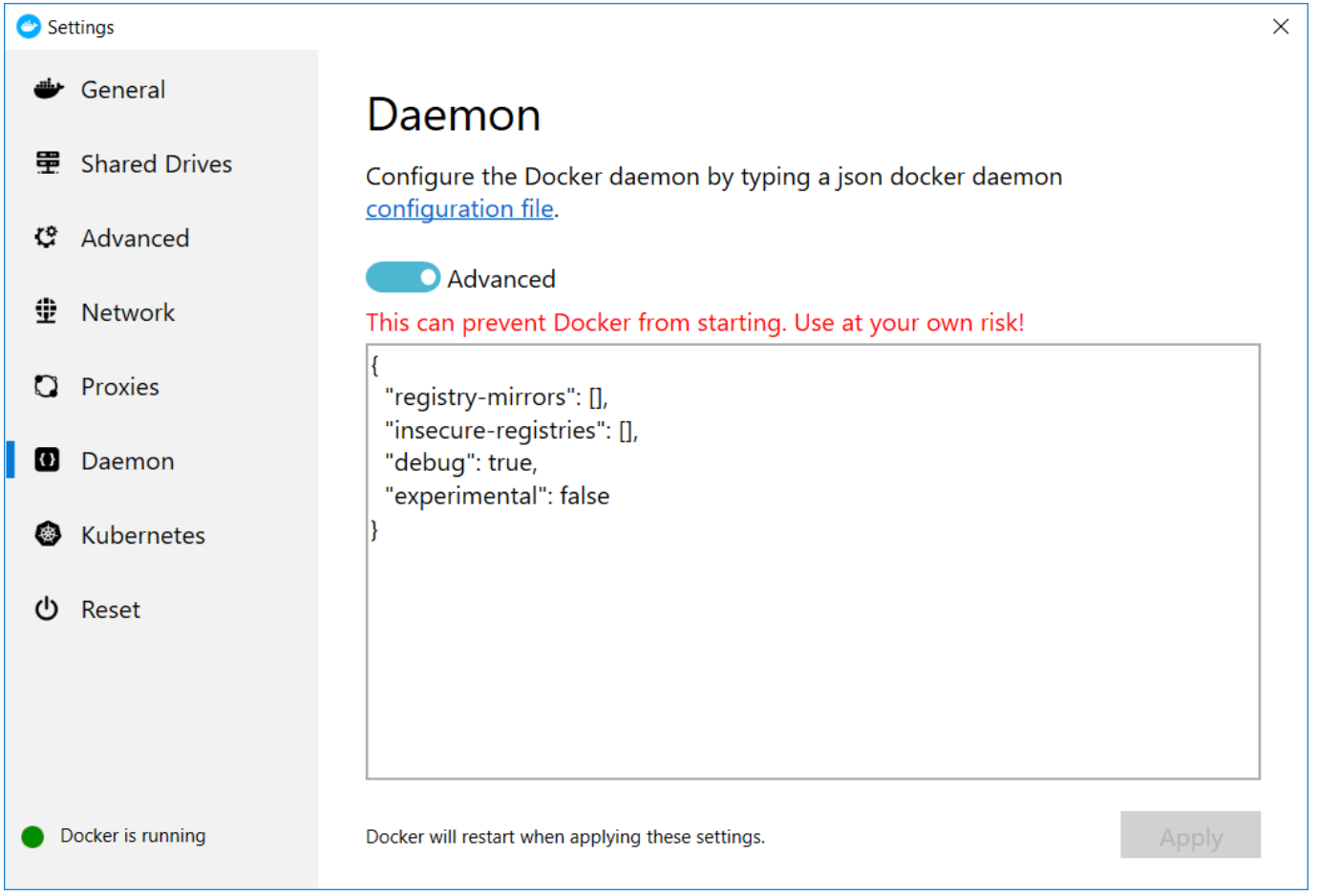
Прокси настроки

Docker Desktop – настройки

Допълнителни настроки
на демона

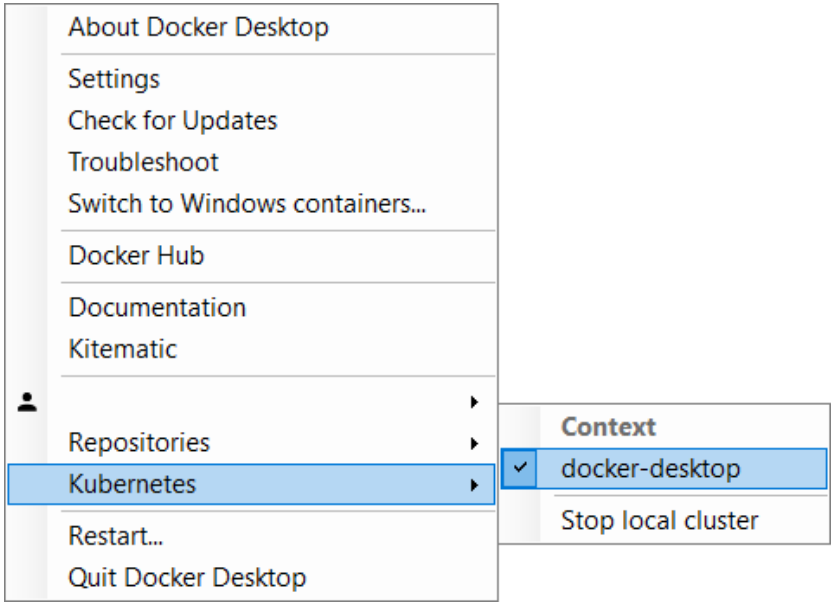


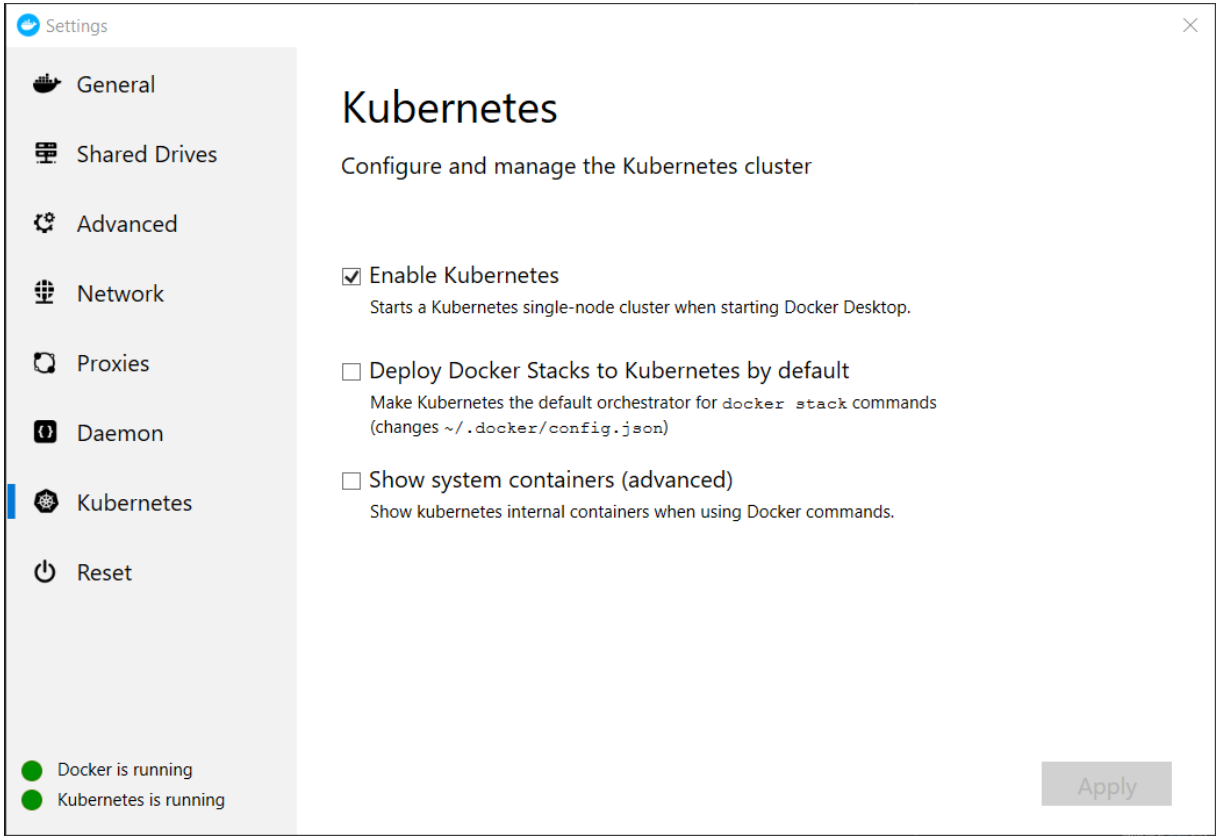
Docker Desktop – настройки



Може да зададем JSON настройки на демона

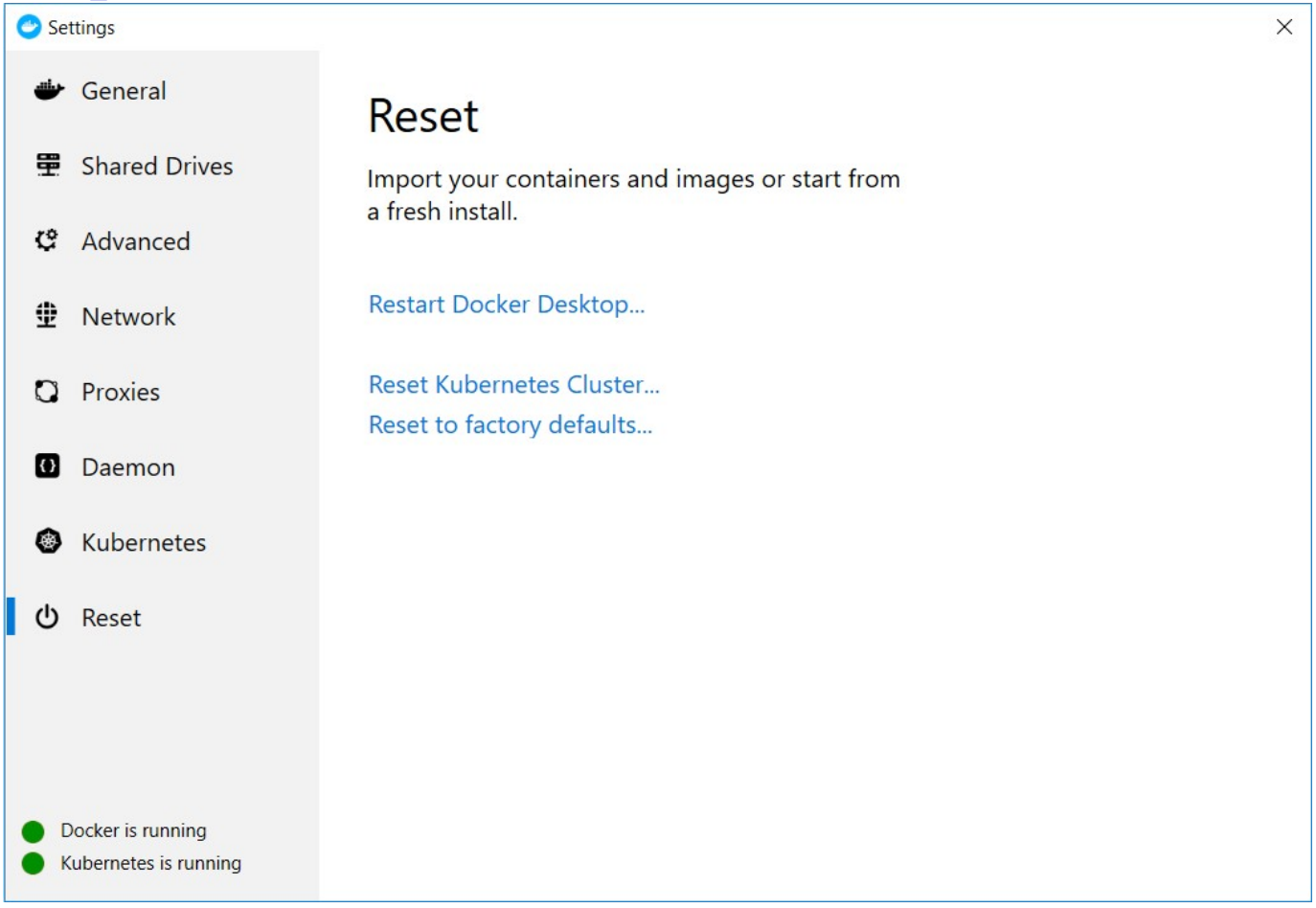
Docker Desktop – настройки





Интеграция на Kubernetes
с Docker

Docker Desktop – настройки



Ако всичко се обърка...

Създаване на Контейнери с Dockerfile

Dockerfile – пример

- Dockerfile съдържа инструкции как да бъде създаден Docker Image;
- **`docker build .`**
- Получаваме Docker Image готов за създаване на инстанции/Контейнери;
- **`docker images`**
- **`docker run ...`**

```
FROM node:9.3.0-alpine

RUN npm install -g @angular/cli@1.5.5 \
     && mkdir -p /usr/src/pintail-whoami

WORKDIR /usr/src/pintail-whoami

ADD . /usr/src/pintail-whoami

RUN npm install && ng build

EXPOSE 80

CMD node server.js $HOSTNAME
```

Docker Compose

Docker Compose

- **Docker Compose** е инструмент за дефиниране и изпълнение на многоконтейнерни Docker приложения;
- Използва се **YAML** файл за конфигуриране на услугите на приложението;
- Една команда върши необходимото: **docker-compose up**

Docker Compose – Работа в три стъпки

- Дефинираме обкръжението на приложението с **Dockerfile**, за да имаме желаната репродуцируемост навсякъде;
- Дефинираме услугите, които изграждат приложението с **docker-compose.yaml**, така че да могат да се изпълняват заедно в изолирана среда;
- Стартираме **docker-compose up** и това стартира цялото приложение;

Пример: Dockerfile и docker-compose.yaml

```
From centos:7


RUN yum -y install --
  setopt=tsflags=nodocs epel-release &&
yum -y install --setopt=tsflags=nodocs
  nginx &&
yum clean all


EXPOSE 80
EXPOSE 443


CMD [ "nginx", "-g", "daemon off;"]
```

```
version: '3'
services:
  nginx:
    image: nginx:latest
    volumes:
    - /opt/ng/nginx-
  configuration:/etc/ngi
  nx
  ports:
    - 80:80
    - 443:443
```

Оркестрация

Оркестрация е автоматизирана конфигурация, координация и управление на комплексни компютърни системи, услуги и софтуер

Инструменти за автоматизация на конфигурирането

Съществуват много инструменти за автоматизация на сървърната конфигурация и управление:

- Puppet;
- Ansible;
- Fabric;
- SaltStack;
- CHEF;
- Terraform;
- AWS CloudFormation;
- и др.







AWS
CloudFormation









HashiCorp
Terraform

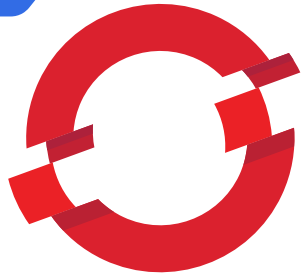
Инструменти за оркестрация на Контейнери

За оркестрация на Контейнери:

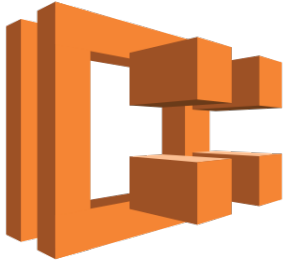
- Kubernetes;
- AWS EKS – Elastic Kubernetes Service;
- AWS ECS – Elastic Container Service;
- Amazon Fargate;
- Kontena;
- OpenShift (базиран на Docker, Kubernetes и др.)
- и др.







OPENSHIFT





Въпроси?

apenev@uni-plovdiv.bg