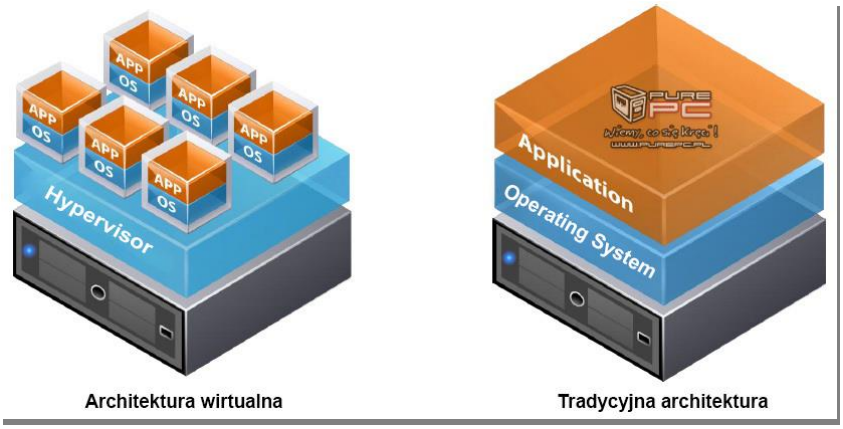
Systemy operacyjne 12

Wirtualizacja

Wirtualizacja polega na utworzeniu symulowanego (wirtualnego) środowiska komputerowego, które stanowi przeciwieństwo środowiska fizycznego. Wirtualizacja często obejmuje wygenerowane komputerowo wersje sprzętu, systemów operacyjnych, urządzeń magazynujących itp. Dzięki temu organizacje mogą podzielić na partycje jeden komputer fizyczny lub serwer, tworząc kilka maszyn wirtualnych. Maszyny wirtualne mogą niezależnie wchodzić w interakcje. Mogą mieć też zainstalowane różne systemy operacyjne i aplikacje, ale współużytkować zasoby jednego komputera-hosta.

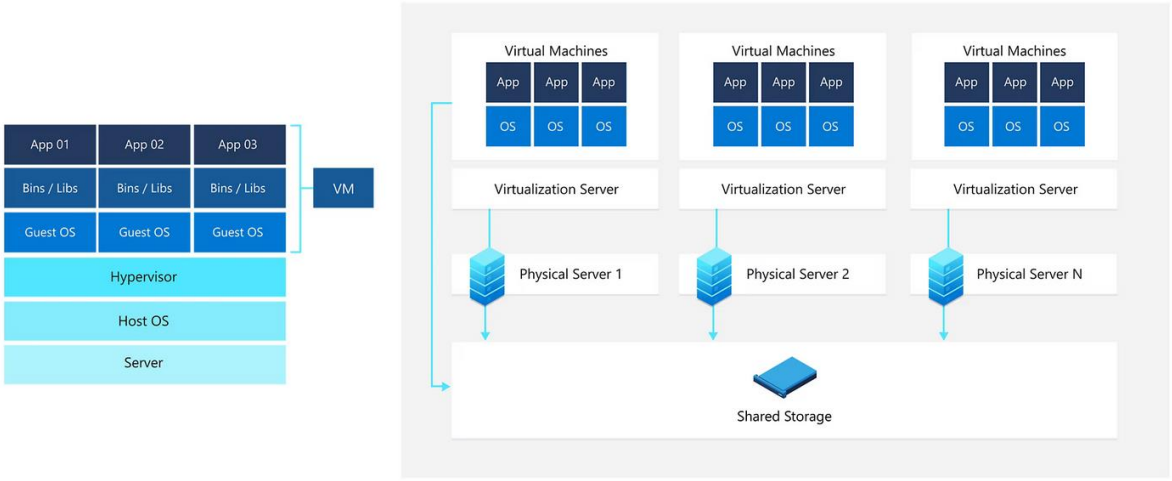
Wirtualizacja umożliwia utworzenie wielu zasobów na bazie jednego komputera lub serwera. W ten sposób poprawia skalowalność i zapewnia obsługę większej liczby obciążeń, a jednocześnie zmniejsza liczbę wymaganych serwerów, ilość zużywanej energii, koszty infrastruktury i nakład prac konserwacyjnych. Istnieją cztery główne kategorie wirtualizacji:

- Pierwsza to wirtualizacja pulpitu. Umożliwia ona zapewnienie i obsługę indywidualnych pulpitów przez jeden centralny serwer.
- Druga to wirtualizacja sieci. Polega ona na podzieleniu przepustowości sieci na niezależne kanały, które można przypisać do określonych serwerów lub urządzeń.
- Trzecia kategoria to wirtualizacja oprogramowania. Oddziela ona aplikacje od sprzętu i systemu operacyjnego.
- Czwarta to wirtualizacja magazynu. Łączy ona wiele zasobów magazynu sieciowego w jedno urządzenie magazynujące dostępne dla wielu użytkowników.



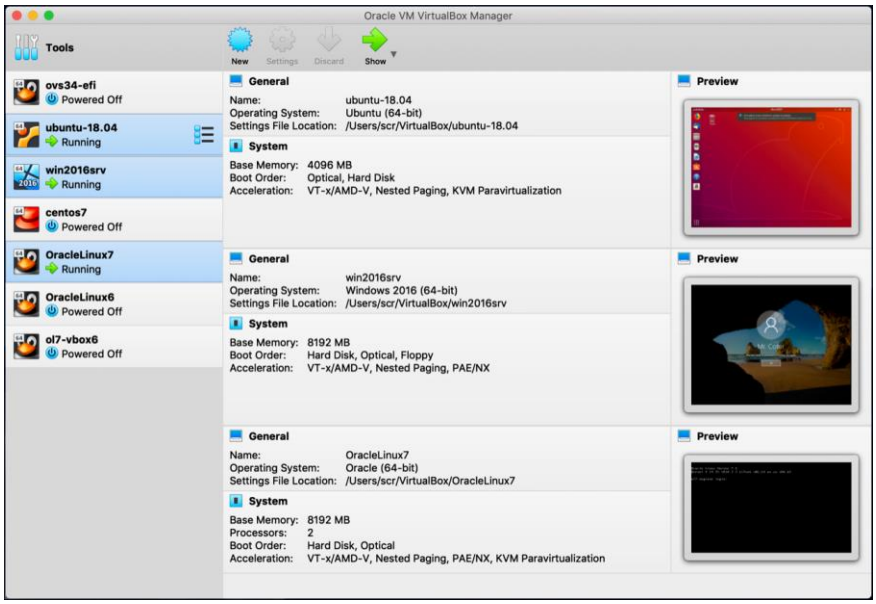
1. Maszyna wirtualna

Maszyna wirtualna nie różni się niczym od każdego innego komputera fizycznego, takiego jak laptop, smartfon lub serwer. Ma procesor CPU, pamięć, dyski do przechowywania plików i jeśli jest to konieczne, może połączyć się z Internetem. Chociaż części tworzące komputer (nazywane sprzętem) są fizyczne i materialne, maszyny wirtualne często są traktowane jako komputery wirtualne lub komputery zdefiniowane programowo w ramach serwerów fizycznych, istniejących tylko jako kod.



VirtualBox

Oracle VM VirtualBox. Oracle VM VirtualBox, **najpopularniejsze na świecie międzyplatformowe oprogramowanie wirtualizacyjne typu open source**, umożliwia programistom szybsze dostarczanie kodu poprzez uruchamianie wielu systemów operacyjnych na jednym urządzeniu.

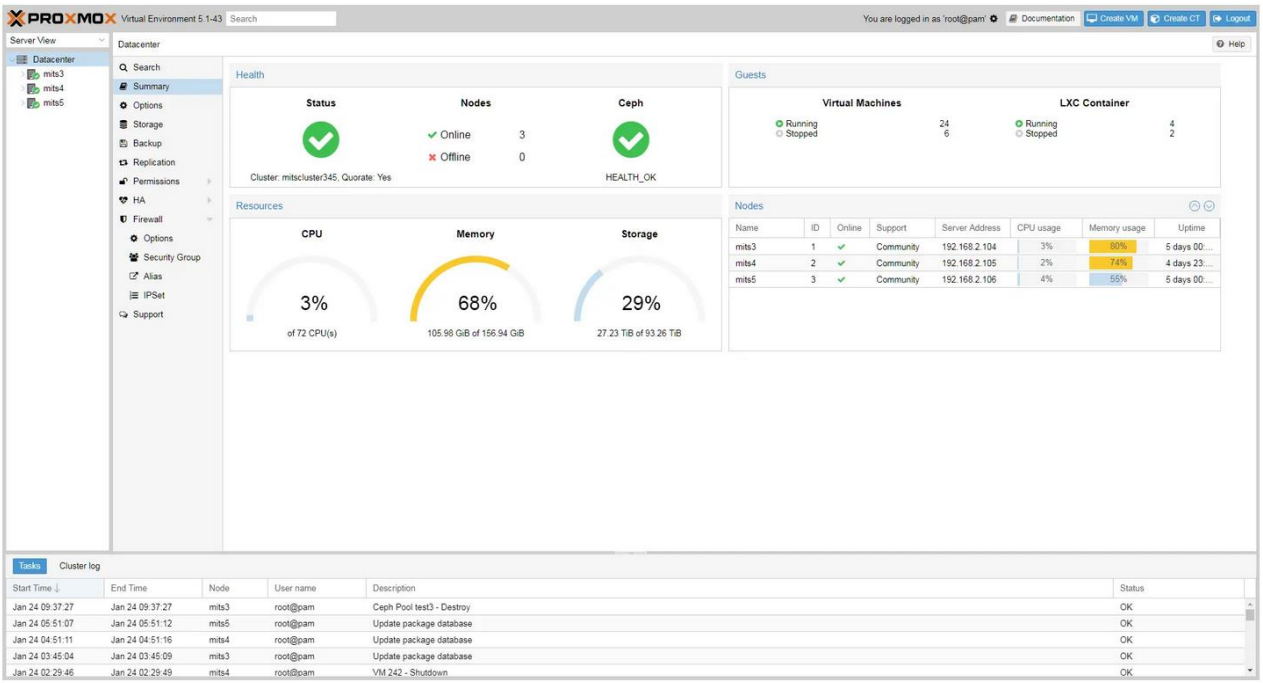


Proxmox

PROXMOX

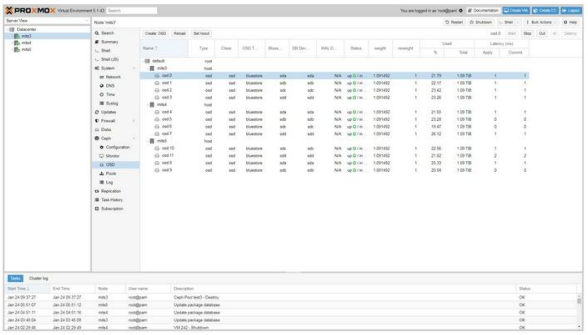
Co to jest Proxmox?

Proxmox jest środowiskiem wirtualizacji działającym w oparciu o system Debian. Od strony sprzętowej wymagana jest 64 bitowa architektura procesora, oraz ewentualne wsparcie wirtualizacyjne – w Intelu nazywane Intel-VT, natomiast w AMD jest to AMD-V.



Model danych

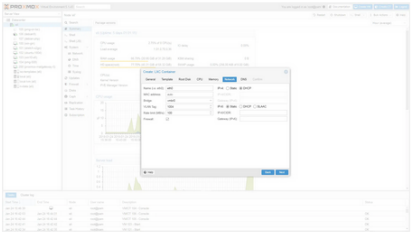
Wirtualizacja Proxmox VE wspiera LVM, katalogi oraz system ZFS, współpracuje również z rozwiązaniami takimi, jak: iSCSI, Fiber Channel, NFS, GlusterFS, CEPH i DRBD.



LXC

Jest to otwarta platforma wirtualizacji systemu operacyjnego dla Linuksa. Pozwala niezależnie od siebie uruchomić kilka systemów wirtualnych (VPS), używających wspólnego jądra z hypervisorem. Mogą to być na przykład różne dystrybucje, środowiska testowe czy serwery dzierżawione. Każde z tych środowisk wirtualnych może być zarządzane zupełnie osobno, tak jakby było fizycznym serwerem. Ograniczeniem jest brak możliwości zmiany jądra, gdyż jest ono wspólne.

Proste zarządzanie LXC, KVM

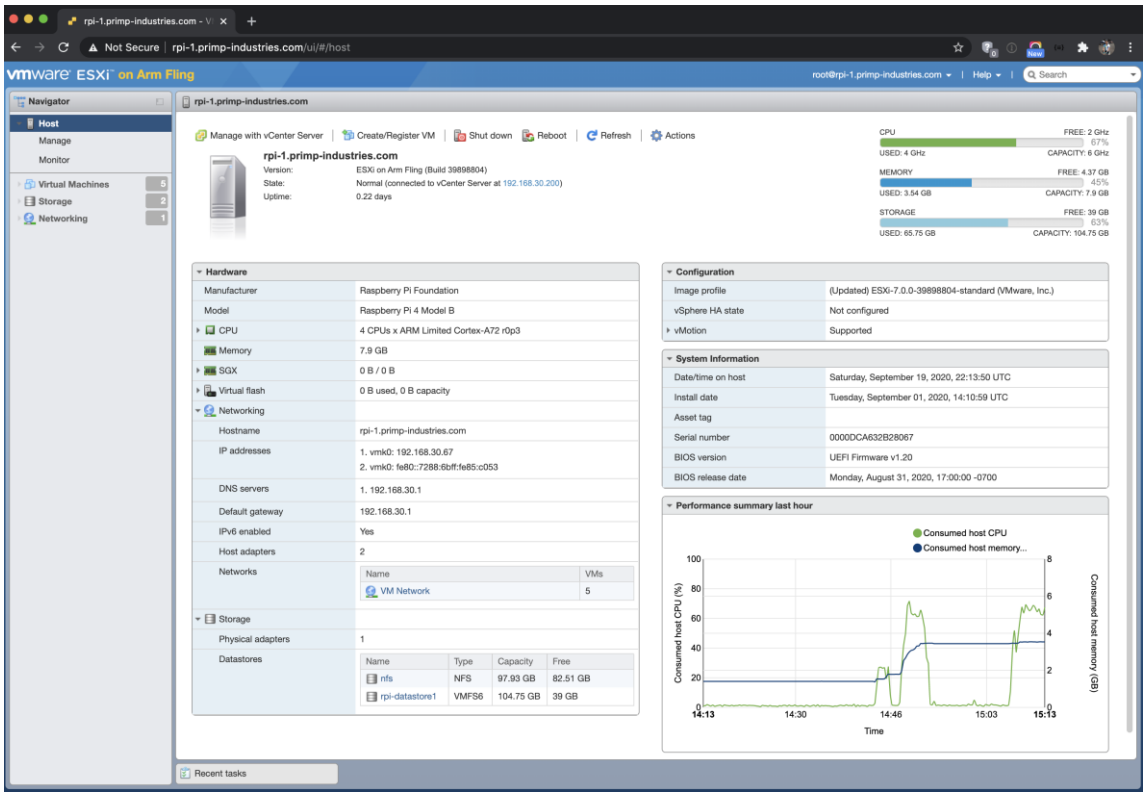


KVM, Qemu

Maszyna wirtualna bazująca na jądrze to środowisko wirtualizacyjne dla systemów linuksowych. Wyróżnia się system gospodarza i gościa. Gospodarz to system operacyjny (OS) kontrolujący sprzęt – „goły metal". Gość to OS wewnątrz gospodarza, któremu gospodarz udostępnia sprzęt. Środowisko KVM (gospodarz lub gospodarze) umożliwia tworzenie sieci maszyn wirtualnych (goście). Każda maszyna wirtualna posiada prywatny zwirtualizowany sprzęt: bios, kartę sieciową, dysk twardy, kartę graficzną, dźwiękowa, porty I/O itd.

VMware

Aplikacja VMware Workstation **pozwala uruchamiać wiele systemów operacyjnych jednocześnie na jednej fizycznej maszynie**. Dodatkowo, systemy uruchamiane jako wirtualne mogą komunikować się między sobą za pomocą protokołów internetowych. Pozwala to np. na testowanie tworzonych rozwiązań klient-serwer.



2. CHROOT

Chroot jest skrótem od *change root* i określa mechanizm zmiany katalogu bazowego dla uruchamianego procesu. Powoduje to, że dany proces oraz wszystkie jego procesy pochodne odnosząc się do ścieżki / odnoszą się do zmienionego katalogu.

Jak działa chroot?

Chroot działa, kopiując aplikację i wszystkie jej pliki wykonywalne i zależności do alternatywnego katalogu głównego. Następnie uruchamia aplikację z tego alternatywnego katalogu głównego, powodując, że aplikacja traktuje ją jako oryginalny katalog główny. Katalog główny jest najwyższym katalogiem w hierarchii i żadna aplikacja nie może sięgać wyżej niż ten katalog, więc w ten sposób chroot izoluje aplikację od reszty systemu.

Przypadków użycia

- Konfigurowanie środowiska testowego
- Uruchamianie programów 32-bitowych w systemie 64-bitowym
- Uruchamianie starszych wersji programu w najnowszej wersji systemu operacyjnego
- Odzyskiwanie hasła

Składnia

Poniżej znajduje się podstawowa składnia polecenia chroot:

$ chroot<ścieżka/do/alternatywny/źródło/informator>Komenda

Jak zrobić pierwszego *chroot*-a?

1. Przygotowanie katalogu i wgranie basha

```
mkdir /tmp/first_chroot
mkdir /tmp/first_chroot/bin
cp /bin/bash /tmp/first_chroot/bin/
```

3. LXC

Główną zaletą LXC jest możliwość **uruchomienia pełnoprawnego systemu operacyjnego w kontenerze**, który uruchamiany jest na współdzielonym jądrze systemu hosta. Jeśli chodzi o zasadę działania, LXC jest bardziej „zbliżone" do maszyn wirtualnych.

O kontenerach LXC możemy myśleć jako czymś zbliżonym do tego, za co w systemie Linux odpowiedzialne jest polecenie **chroot**. Chroot to polecenie uniksowe, pozwalające uruchomić dany program ze zmienionym korzeniem (root) – katalogiem głównym systemu plików.

Instalacja

apt install lxc-utils

Tworzenie kontenerów LXC

Po zainstalowaniu niezbędnych pakietów, możemy przystąpić do tworzenia kontenerów LXC. Pierwsza możliwość to stworzenie kontenera w trybie interaktywnym. Po wpisaniu poniższego polecenia, zostaniemy poproszeni o wskazanie:

- Dystrybucji
- Wersji
- Architektury

```
$ sudo lxc-create --template download --name u1

Distribution:
ubuntu
Release:
bionic
Architecture:
amd64

Downloading the image index
Downloading the rootfs
Downloading the metadata
The image cache is now ready
Unpacking the rootfs

---
You just created an Ubuntu bionic amd64 (20200328_07:42) container.
```

Ten sam efekt możemy uzyskać jednym poleceniem:

```
$ lxc-create -t download -n u1 -- --dist ubuntu --release bionic --arch amd64
```

Do wyświetlanie kontenerów służy polecenie `lxc-ls`

```
$ lxc-ls --fancy
NAME STATE   AUTOSTART GROUPS IPV4 IPV6 UNPRIVILEGED
u1   STOPPED 0         -      -    -    false
```

Przydatne polecenia

lxc image list

lxc launch images:ubuntu/18.04 first

lxc stop first

lxc delete first

lxc list

lxc config show first

lxc exec first — /bin/bash

4. Vagrant

Oprogramowanie typu open source do tworzenia i utrzymywania przenośnych wirtualnych środowisk programistycznych; np. dla VirtualBox, KVM, Hyper-V, kontenerów Docker, VMware, Parallels i AWS.

Instalacja Vagrant-a i VirtualBox-a

Do działania Vagrant-a potrzebujemy tzw. hipernadzorcę jakim jest np. VirtualBox. Do dyspozycji mamy także Hyper-V, Docker, VMware oraz AWS. Jeśli pobierzecie instalator ze strony Vagrant-a to powinniście mieć VirtualBox-a zawartego w instalatorze.

Dodatkowo warto w przypadku użytkowników Windows-a zainstalować dodatkowe oprogramowanie umożliwiające łączenie się poprzez protokół ssh. Jest to potrzebne do zalogowania się do wirtualnej maszyny, którą będzie tworzył Vagrant. Osobiście polecam putty, które jest spoko, ale genialnym rozwiązaniem jest zainstalowanie sobie GIT-a. Wraz z GIT-em zostanie zainstalowane MinGW (Minimalist GNU for Windows), dzięki któremu zyskujemy dostęp do poleceń typowo linuks-owych w tym ssh.

Konfiguracja maszyny wirtualnej – Vagrantfile

Do konfiguracji maszyny wirtualnej wykorzystywany jest plik Vagrantfile, który opisuje konfigurację maszyny. Plik ten jest plikiem tekstowym, którego składnia została oparta o język Ruby (znajomość języka Ruby nie jest wymagana).

Przykładowy plik Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
```

```
# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented
below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search
for
  # boxes at https://atlas.hashicorp.com/search.
  config.vm.box = "base"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example
below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # NOTE: This will enable public access to the opened port
  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine and only allow
access
  # via 127.0.0.1 to disable public access
  # config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip:
"127.0.0.1"

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  # config.vm.network "private_network", ip: "192.168.33.10"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network "public_network"

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  # config.vm.synced_folder "../data", "/vagrant_data"

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  # config.vm.provider "virtualbox" do |vb|
  #   # Display the VirtualBox GUI when booting the machine
  #   vb.gui = true
  #
  #   # Customize the amount of memory on the VM:
  #   vb.memory = "1024"
  # end
```

```
  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Define a Vagrant Push strategy for pushing to Atlas. Other push
strategies
  # such as FTP and Heroku are also available. See the documentation at
  # https://docs.vagrantup.com/v2/push/atlas.html for more information.
  # config.push.define "atlas" do |push|
  #   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
  # end

  # Enable provisioning with a shell script. Additional provisioners such
as
  # Puppet, Chef, Ansible, Salt, and Docker are also available. Please see
the
  # documentation for more information about their specific syntax and use.
  # config.vm.provision "shell", inline: &amp;amp;lt;&amp;amp;lt;-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

5. Docker

Docker jest platformą, dzięki której możemy rozwijać oraz wdrażać nasze aplikacje w postaci samowystarczalnych kontenerów. Kontenery są odizolowanym środowiskiem uruchomieniowym dla naszej aplikacji. Można porównać je do maszyn wirtualnych uruchamianych na Windowsie np. za pomocą Hyper-V. Dzięki temu możemy na jednym fizycznym sprzęcie uruchomić Windowsa oraz Linuxa. Jednak każda maszyna wirtualna posiada własny system operacyjny. Tym samym uruchomienie kilku maszyn oznacza uruchomienie kilku systemów operacyjnych, co może być dużym obciążeniem dla jednego komputera. Do tego najczęściej nowo uruchomiony system należy odpowiednio skonfigurować i zainstalować wszystkie potrzebne nam biblioteki w odpowiednich wersjach i konfiguracjach. Tak naprawdę interesujące i przydatne dla nas jest to wszystko, co działa wewnątrz systemu operacyjnego, a nie sam OS.

Docker i kontenery znacznie upraszczają cały ten proces. Zamiast instalować hipervisora i całe systemy operacyjne uruchamiamy jedynie to, co powinno działać *wewnątrz* tego systemu (czyli np. NodeJS bądź MongoDB). Jedynym systemem operacyjnym jest wtedy OS działający na naszym komputerze-hoście. Docker jest odpowiedzialny za to, żeby *podkraść* z naszego fizycznego sprzętu nieco zasobów (tj. miejsce na dysku twardym, RAM, usługi sieciowe itp.) i przeznaczyć je dla pojedynczego kontenera. A co najważniejsze – to co zostanie przydzielone kontenerowi, należy tylko do tego kontenera. Nawet host nie ma dostępu do zasobów, z których korzysta kontener. Wszystkie kontenery są odizolowane zarówno od siebie, jak i od naszego systemu operacyjnego

Co i jak powinno być uruchomione wewnątrz kontenera, jest określone przez plik nazywany *Dockerfile*. To tam znajdują się informacje, jakie wersje biblioteki powinny być zainstalowane, jak powinny zostać skonfigurowane, w jakiej kolejności uruchomione itp.

Przykładowy plik Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
```

```
CMD ["node", "src/index.js"]
EXPOSE 3000
```

docker-compose

Kontenery dockerowe można uruchamiać bezpośrednio z konsoli, stosując polecenie docker run. Jest to jednak niepraktyczne - często chcemy wykonać to polecenie z wieloma parametrami, co sprawia, że staje się ono tasiemcowate i podatne na pomyłki. Nie wspominam w ogóle o sytuacji, kiedy chcemy uruchomić kilka współpracujących ze sobą kontenerów... Na pomoc przychodzi docker-compose. Jest to wrapper na dockera, czyli wykonuje ostatecznie dokładnie te same czynności, które wykonałyby się, jakbyśmy to wpisali z palca, jednak bazuje on nie na parametrach z konsoli, a na opisie stacka dockerowego w pliku YML. Dzięki temu nie musimy za każdym razem pamiętać całego zestawu poleceń, plik możemy trzymać w repozytorium i dzięki temu łatwo współdzielić go w ramach projektu i wersjonować, a sam plik jest w YAMLu, więc jest czytelny, co z kolei daje nam informację o samym stacku, który uruchamiamy - bez konieczności zrozumienia wielolinijkowych tasiemców.

Struktura pliku docker-compose.yml

```
version: "3.7"
services:

  nginx-proxy:
    image: jwilder/nginx-proxy:latest
    restart: always
    ports:
      - "80:80"
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro

  mysql:
    image: mysql:5.7
    restart: always
    container_name: vershold_db
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_USER: app
      MYSQL_PASSWORD: app
      MYSQL_DATABASE: app

  php:
    build: containers/php
    restart: always
    container_name: vershold_php
    depends_on:
      - mysql
    volumes:
      - ./code:/var/www/magento
    expose:
      - 9000

  nginx:
    build: containers/nginx
    restart: always
    container_name: vershold_nginx
    depends_on:
      - php
```

```
volumes:
  - ./code:/var/www/magento
expose:
  - 80
environment:
  - VIRTUAL_HOST=vershold.127.0.0.1.xip.io
```

Instalacja dockera

#Docker

apt -yf remove docker docker-engine docker.io containerd runc

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

add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb_release -cs) \

stable"

apt update

apt install -yf docker-ce docker-ce-cli containerd.io

systemctl enable docker

groupadd docker

#Docker-compose

sudo curl -L "https://github.com/docker/compose/releases/download/1.28.6/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

Na koniec należy dodać swojego użytkownika do grupy docker

sudo usermod -aG docker nazwa_usera

6. Warden

Jest to nakładka na docker-compose z dodatkowo skonfigurowanymi usługami pozwalającego na szybkie uruchomienie systemów takich jak magento, shopware, drupal lub swojego autorskiego projektu. Warden był stosowany do macOs ze względu na jego wydajność gdy zwykły docker był bardzo problematyczny.

1. Utwórz nowy katalog na komputerze hosta w wybranej lokalizacji, a następnie przejdź do nowego katalogu, aby rozpocząć:

```
mkdir -p ~/Sites/exampleproject
cd ~/Sites/exampleproject
```

2. Z katalogu głównego nowego katalogu projektu uruchom, env-initaby utworzyć .envplik z konfiguracją potrzebną Wardenowi i Dockerowi do pracy z projektem.

```
warden env-init exampleproject drupal
```

3. Wynikiem tego polecenia jest .env plik w katalogu głównym projektu (wskazówka: zatwierdź to w VCS, aby udostępnić konfigurację innym członkom zespołu) o następującej zawartości:

```
WARDEN_ENV_NAME=exampleproject
WARDEN_ENV_TYPE=drupal

WARDEN_WEB_ROOT=/

TRAEFIK_DOMAIN=tdpl.test
TRAEFIK_SUBDOMAIN=app
DB_DISTRIBUTION=mariadb
DB_DISTRIBUTION_VERSION=10.4
NODE_VERSION=18
COMPOSER_VERSION=2
PHP_VERSION=8.2
PHP_XDEBUG_3=1

WARDEN_DB=1
WARDEN_RABBITMQ=0
WARDEN_REDIS=0

RABBITMQ_VERSION=3.8

WARDEN_SYNC_IGNORE=

WARDEN_ALLURE=0
WARDEN_SELENIUM=0
WARDEN_SELENIUM_DEBUG=0
WARDEN_BLACKFIRE=0

BLACKFIRE_CLIENT_ID=
BLACKFIRE_CLIENT_TOKEN=
BLACKFIRE_SERVER_ID=
BLACKFIRE_SERVER_TOKEN=
```

4. Podpisz certyfikat SSL do użytku z projektem (dane wejściowe powinny być zgodne z wartością TRAEFIK_DOMAINw powyższym .env przykładowym pliku):

```
warden sign-certificate exampleproject.test
```

5. Następnie będziesz chciał uruchomić środowisko projektowe:

```
warden env up
```

6. Polecenia następujące po tym kroku w procedurze konfiguracji zostaną uruchomione z poziomu php-fpm kontenera dokera, do którego zostaniesz uruchomiony:

```
warden shell
```

Zadania do wykonania

1. Poszukaj informacji o wadach i zaletach dockera i lxc
2. Poszukaj innych metod wirtualizacji