1. Docker - основні поняття та команди:

Docker - це платформа для розробки, доставки та запуску додатків у контейнерах. Основні поняття включають:

- Образ (Image): шаблон для створення контейнера

- Контейнер (Container): ізольоване середовище для запуску додатку

- Dockerfile: файл з інструкціями для створення образу

- Docker Hub: репозиторій для зберігання та обміну образами

Основні команди:

- docker run: запуск контейнера

- docker build: створення образу

- docker pull: завантаження образу

- docker push: відправка образу в репозиторій

- docker ps: перегляд запущених контейнерів

2. Відмінність між Docker IMAGE та Docker CONTAINER:

- Image (образ) - це статичний шаблон з кодом, залежностями та конфігурацією для створення контейнера.

- Container (контейнер) - це запущений екземпляр образу, який виконується в ізольованому середовищі.

3. Тег ALPINE в докер образах:

Alpine - це мінімалістичний дистрибутив Linux. Тег ALPINE означає, що образ базується на Alpine Linux, що робить його легким та компактним.

4. Перегляд зупинених контейнерів:

Для перегляду всіх контейнерів, включаючи зупинені, використовується команда:

```

docker ps -a

```

5. Роль Dockerfile при створенні образів:

Dockerfile - це текстовий файл з інструкціями для автоматичного створення Docker-образу. Він визначає базовий образ, копіює файли, встановлює залежності та налаштовує середовище для роботи додатку.

1. Docker – інструкції Dockerfile:

Основні інструкції Dockerfile включають:

- FROM: вказує базовий образ

- COPY: копіює файли з хост-системи в образ

- ADD: копіює файли, але також може розпаковувати архіви

- RUN: виконує команди під час створення образу

- CMD: задає команду за замовчуванням для запуску контейнера

- ENTRYPOINT: визначає головну команду контейнера

- ENV: встановлює змінні середовища

- EXPOSE: інформує Docker про порти, які слухає контейнер

- VOLUME: створює точку монтування для постійного зберігання даних

- WORKDIR: встановлює робочу директорію для наступних інструкцій

2. Прокидання портів при запуску докер контейнерів:

Прокидання портів - це механізм, який дозволяє зв'язати порт на хост-машині з портом всередині контейнера. Це потрібно для того, щоб зовнішні програми могли взаємодіяти з сервісами, що працюють у контейнері. Наприклад, якщо веб-сервер у контейнері слухає порт 80, ми можемо прокинути його на порт 8080 хост-машини, щоб отримати доступ до сайту ззовні.

3. VOLUME у докері:

VOLUME - це механізм для створення постійного сховища даних, яке існує незалежно від життєвого циклу контейнера. Це дозволяє зберігати дані навіть після видалення контейнера, а також ділитися даними між контейнерами. VOLUME корисний для баз даних, конфігураційних файлів та інших даних, які потрібно зберігати постійно.

4. Інструкція RUN у Dockerfile:

RUN виконує команди в новому шарі поверх поточного образу і фіксує результати. Зазвичай використовується для встановлення пакетів, компіляції коду, створення файлів і директорій. Кожна інструкція RUN створює новий шар в образі.

5. Команда для входу в командний рядок контейнера Docker:

Для входу в командний рядок працюючого контейнера використовується команда:

docker exec -it [ID\_контейнера] /bin/bash

або

docker exec -it [ім'я\_контейнера] /bin/bash

Де -it забезпечує інтерактивний режим з підключенням до терміналу, а /bin/bash - це команда, яка буде виконана всередині контейнера (в даному випадку - запуск оболонки bash).

Docker Volume - це механізм для зберігання даних поза контейнером, що забезпечує постійність даних та полегшує обмін даними між контейнерами. Ось детальніша інформація та приклади:

1. Типи Volume:

- Named volumes: створені та керовані Docker

- Host volumes: прив'язані до конкретної директорії на хост-системі

- Anonymous volumes: створені Docker, але без вказаного імені

2. Переваги використання Volume:

- Дані зберігаються після видалення контейнера

- Легко переносити дані між контейнерами

- Покращена продуктивність порівняно з використанням файлової системи контейнера

- Можливість резервного копіювання або міграції даних

3. Приклади використання:

a) Створення та використання named volume:

# Створення volume

docker volume create my\_data

# Запуск контейнера з використанням volume

docker run -d --name my\_container -v my\_data:/app/data my\_image

b) Використання host volume:

docker run -d --name my\_container -v /path/on/host:/app/data my\_image

c) Створення volume через Dockerfile:

FROM ubuntu

VOLUME /app/data

d) Приклад з базою даних (PostgreSQL):

# Створення volume для даних PostgreSQL

docker volume create pgdata

# Запуск контейнера PostgreSQL з використанням volume

docker run -d --name my\_postgres -v pgdata:/var/lib/postgresql/data -e POSTGRES\_PASSWORD=mysecretpassword postgres

4. Команди для роботи з Volume:

- Створення: `docker volume create my\_volume`

- Список всіх volume: `docker volume ls`

- Інспектування volume: `docker volume inspect my\_volume`

- Видалення: `docker volume rm my\_volume`

- Видалення невикористаних volume: `docker volume prune`

5. Приклад використання volume для обміну даними:

# Створення volume

docker volume create shared\_data

# Запуск першого контейнера, який записує дані

docker run -it --name writer -v shared\_data:/data ubuntu bash

# В контейнері: echo "Hello from container 1" > /data/message.txt

# Запуск другого контейнера, який читає дані

docker run -it --name reader -v shared\_data:/data ubuntu bash

# В контейнері: cat /data/message.txt

6. Резервне копіювання даних з volume:

docker run --rm -v my\_volume:/source -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /source

Використання Docker Volume є важливою практикою для забезпечення постійності даних та ефективного управління станом у контейнеризованих додатках.

**3333333333333**

1. Провайдери віртуальних машин:

Провайдери віртуальних машин - це програмні платформи, які дозволяють створювати та керувати віртуальними машинами. Приклади:

- VirtualBox: безкоштовний, кросплатформенний провайдер від Oracle.

- VMware: комерційне рішення з розширеними функціями для підприємств.

- Hyper-V: вбудований у Windows провайдер від Microsoft.

- KVM: відкритий провайдер для Linux-систем.

- AWS EC2: хмарний провайдер від Amazon для створення віртуальних машин в хмарі.

2. Vagrant Boxes та Vagrant Cloud:

- Vagrant Boxes: це попередньо налаштовані образи віртуальних машин для Vagrant.

- Vagrant Cloud: онлайн-платформа для зберігання та обміну Vagrant Boxes.

Користувачі можуть завантажувати готові boxes або публікувати власні для спільного використання.

3. Vagrantfile - ініціалізація та інструкції:

Vagrantfile - це конфігураційний файл для Vagrant, написаний на Ruby. Основні інструкції:

```ruby

Vagrant.configure("2") do |config|

config.vm.box = "ubuntu/bionic64" # Вибір базового образу

config.vm.network "private\_network", ip: "192.168.33.10" # Налаштування мережі

config.vm.provider "virtualbox" do |vb|

vb.memory = "1024" # Налаштування пам'яті

end

config.vm.provision "shell", path: "bootstrap.sh" # Скрипт для початкової конфігурації

end

```

4. Команди Vagrant:

- `vagrant init`: Ініціалізація нового Vagrantfile

- `vagrant up`: Створення та запуск віртуальної машини

- `vagrant ssh`: Підключення до віртуальної машини через SSH

- `vagrant halt`: Зупинка віртуальної машини

- `vagrant destroy`: Видалення віртуальної машини

- `vagrant status`: Перевірка статусу віртуальних машин

5. SCM (System Configuration Management):

SCM - це інструменти для автоматизації налаштування та управління серверами.

- Ansible: використовує YAML для конфігурації, не потребує агента на цільових серверах.

- Chef: використовує Ruby DSL, має клієнт-серверну архітектуру.

- Puppet: декларативна мова конфігурації, підходить для великих інфраструктур.

Terraform - це інструмент для автоматизації створення та управління інфраструктурою в різних хмарних провайдерах (AWS, Azure, GCP тощо).

Картинка, яку ви згадали, ймовірно, ілюструє процес розгортання та конфігурації інфраструктури за допомогою Vagrant та одного з інструментів SCM. Типовий процес може виглядати так:

1. Vagrant створює віртуальну машину на основі вказаного box'а.

2. Після створення VM, Vagrant може використовувати провізіонер (наприклад, Ansible) для налаштування VM.

3. Ansible (або інший SCM-інструмент) застосовує конфігурації, встановлює програмне забезпечення та налаштовує сервіси на VM.

Це дозволяє автоматизувати весь процес від створення VM до її повного налаштування для роботи, забезпечуючи відтворюваність та масштабованість інфраструктури.