Container Manager

Virtual machine membutuhkan menginstall Sistem Operasi, sehingga resourcenya banyak. Setiap Virtual Machine berjalan di Hypervisor seperti Virtual Box

Container berjalan fokus pada sisi aplikasi, sehingga tidak perlu menginstall Sistem Operasi. Container tidak secara langsung berjalan di sistem operasi, melainkan di Container Manager seperti **DOCKER**.

Arsitektur Docker: Client-Server (daemon)

Docker Client dan Docker Daerom berkomunikasi dengan REST API

Docker Registry: Tempat menyimpan docker image.

Contoh Docker Registry:

* Docker Hub
* Digital Ocean Container Registry
* Google Cloud Container Registry
* Amazon Elastic Container Registry
* Azure Container Registry

Docker Image: Seperti Installer aplikasi, di dalamnya terdapat aplikasi dan dependency

Perintah docker image:

**Melihat list docker image:**

docker image ls

**Mendowload docker image redis dengan tag latest**

docker image pull redis:latest

**Menghapus docker image redis**:

docker image rm redis:latest

Docker Container: Mirip seperti aplikasi hasil installernya

1 Docker image bisa digunakan untuk beberapa Docker Container, asalkna nama Docker Containernya berbeda

Docker Image tidak bisa diihapus jika sudah membuat Docker Container

Meskipun menggunakan PORT yang sama di beberapa container, tidak akan mempengaruhi PORT pada sistem operasi, dikarenakan PORT tersebut berasal dari PORT Container yang terisolasi, sehingga tidak memengaruhi PORT sistem Operasi maupun container lain.

**Melihat semua Docker Container, baik yang running maupun tidak:**

docker container ls -a

**Melihat Docker Container yang running**

docker container ls

**Membuat Container:**

docker container create –name namacontainer namaimage:tag

**Menjalankan container**

docker container start id container/namacontainer

**Mnghentikan container**

docker container stop id container/namacontainer

**Menghapus container ketika container sudah berhenti:**

docker container rm containerId/namacontainer

**Container LOG**

**Melihat Container Log aplikasi:**

docker container logs containerId/namacontainer

**Melihat Container Log aplikasi secara realtime (-f : dtunggu sampai muncul logi berikutnya) :**

docker container logs -f containerId/namacontainer

**Container Exec**

Fitur ini digunakan untuk mengeksekusi kode program di dalam container atau juga bisa masuk ke dalam container

**Masuk ke container, mengeksekusi program bash script di dalam container dengan Container Exec:**

docker container exec -i -t containerId/namacontainer /bin/bash

* -i : argumen interaktif menjaga input tetap aktif (apa yang kita kirim/inputkan akan dikirimkan ke /bin/bash)
* -t : argumen untuk alokasi pseudo-TTY (terminal akses) (masuk ke container dengan mode terminal)

Container PORT

PORT pada container terisolasi

PORT Forwarding:

Meneruskan sebuah port yang terdapat di sistem Host/ Operasi nya ke dalam Docker Container.

**Melakukan Port Forwarding:**

docker container create --name namacontainer –publish posthost:portcontainer image:tag

* posthost: isi dengan port berapa (bebas), jadi ketika mengakses port posthost di localhost nanti akan diforward ke portcontainer yang ada di dalam container
* port forwarding lebih dari satu, tambahkan dua kali parameter –publish
* --publish disingkat -p

**Container Environment Variable**

Environment Variable : teknik agar konfigurasi applikasi bisa diubah secara dinamis

Menambah Environment Variable:

Dengan perintah --env atau -e:

Docker container create --name namacontainer --env KEY=“value” --env KEY2=“value” image:tag

**Container Stats**

Untuk melihat detail penggunaan resource untuk setiap container:

docker container stats

Container Resource Limit

**Memory:**

ketika membuat container, bisa ditentukan jumlah memory yang digunakan oleh container ini, dengan perintah –memory diikuti dengan angka memory

* B (bytes), k (kilo bytes), m (mega bytes), atau g (giga bytes), misal 100m artinya 100 mega bytes.

CPU:

Mengatur CPU dalam container dengan parameter --cpus

Contoh set dengan nilai 1.5m artinya container bisa menggunakan satu dan setengah CPU core

**Bind Mounts**

Kemampuan mounting (sharing) file atau folder yang terdapat di sistem host/ operasi ke container di docker atau sebaliknya dari container ke sistem host/ operasi.

Seperti ingin menyimpan file database di container ke dalam folder di sistem operasi.

Dengan menggunakan perintah --mount

Parameter Mount:

* Type: bind or volume
* Source: lokasi file atau folder di sistem host
* Destination: Lokasi file atau folder di container
* Readonly: hanya bisa dibaca container, tidak bisa ditulis

Contoh melakukan mounting:

docker container create --name namacontainer --mount “type=bind, source=folder, destination=folder, readonly” image:tag

**Docker Volume**

Di versi terbaru docker direkomendasikan menggunakan Docker Volume daripada Fitur Bind Mounts.

Mirip Bind Mounts, bedanya adalah terdapat management Volume. Anggap saja volume ini seperti storage atau hardisk kosong, yang nanti volume ini dimasukkan ke dalam container, sehingga container bisa menyimpan data ke dalam volume/storage yang kosong ini.

**Membuat Volume:**

docker volume create namavolume

Menghapus Volume:

Pastikan stop dulu container sebelum menghapus:

docker volume rm namavolume

**Container Volume**

Volume yang sudah dibuat, bisa digunakan di container, jika container dihapus, data akan tetap aman di volume.

Cara menggunakan volume di container sama dengan menggunakan bind mount, dengan parameter –mount, namun dengan type volume dan source nama volume

Tahapan:

1. Create Volume

docker volume create mongodata

1. Create container

docker container create --name mongovolume --mount “type=volume, source=mongodata, destination=/data/db” –publish 27019:27017 --env MONGO\_INITDB\_ROOT\_USERNAME=faisal --env MONGO\_INITDB\_ROOT\_PASSWORD=faisal mongo:latest

1. Start Container

docker container start mongovolume

**Backup Volume**

Belum ada cara otomatis melakukan backup volume

Namun bisa memanfaatkan container baru untuk backup data di dalam volume ke dalam archive seperti zip atau tar.gz, lalu archive disimpan di dalam bind mount.

Tahapan Backup

* Matikan container yang menggunakan volume yang akan dibackup
* Buat container baru dengan dua mount (--mount), volume yang akan dibackup, dan bind mount folder dari sistem host/operasi
* Backup menggunakan container dengan cara meng-archive isi volume, dan disimpan di bind mount folder
* Isi file backup sekarang berada di folder sistem host/operasi
* Delete Container yang kita gunakan untuk melakukan backup

Menjalankan Container secara langsung

**Restore Volume**

Setelah backup volume kedalam file archive, file tersebut bisa disimpan ke tempat yang lebih aman seperti cloud storage, disini akan direstore ke volume baru, memastikan data backup tidak corrupt.

Tahapan Restore Volume:

* Buat volume baru untuk lokasi restore data backup
* Buat container baru dengan dua mount, yaitu volume baru untuk restore backup, dan bind mount folder dari sistem host yang berisi file backup
* Restore menggunakan container dengan cara mengekstrak isi backup file ke dalam volume
* Isi file backup sekarang sudah di restore ke volume
* Delete container yang kita gunakan untuk melakukan restore Volume baru yang berisi data backup siap digunakan oleh container baru

**Docker Network**

Secara default container akan saling terisolasi satu sama lain, jadi bisa dipatika tidak bisa memanggil antar container.

Dengan Network, container bisa dikoneksikan dengan container lain dalam satu network yang sama sehingga bisa saling berkomunikasi.

Network Driver

* Bridge: driver untuk membuat network secara virtual, yang memungkinkan container yang terkoneksi di bridge network yang sama saling berkomunikasi.
* host: driver untuk membuat network yang sama dengan sistem host/operasi, seakan akan containernya dikonekan langsung ke sistem hostnya. Host hanya jalan di Docker Linux
* none: drive untuk membuat network yang tidak bisa berkomunikasi

**Melihat Network:**

docker network ls

**Membuat Network tanpa (--drive) defaultnya bridge:**

docker network create --driver namadriver namanetwork

**Menghapus Network:**

docker network rm namanetwork

Network tidak bisa dihapus jika sudah digunakan oleh container. Hapus terlebih dahulu container dari network

**Container Network**

Container bisa mengakses container lain dengan menyebutkan hostname dari container nya, yaitu nama container nya.

Membuat container dengan Network

docker container create --name namacontainer –network namanetwork image:tag

Menghapus Container dari Network

docker network disconnect namanetwork namacontainer

Menambahkan Container ke Network

docker network connect namanetwork namacontainer

**Inspect**

Untuk melihat detail dari setiap image, network, volume dan container.

**Melihat detail image:**

docker image inspect namaimage

**Melihat detail container:**

docker container inspect namacontainer

**Melihat detail volume:**

docker volume inspect namavolume

**Melihat detail network:**

docker network inspect namanetwork

**Prune**

Membersihkan hal-hal yang sudah tidak digunakan.

**Menghapus container yang sudah stop:**

docker container prune

**Menghapus semua image yang tidak digunakan container:**

docker image prune

**Menghapus network yang tidak digunakan container:**

docker network prune

**Menghapus volume yang tidak digunakan container:**

docker volume prune

**Menghapus semua container, image, network, volume yang tidak digunakan:**

docker system prune