**PCC CLASS**

**DEPARTEMEN NETWORK**

**Introduction Container and Docker**

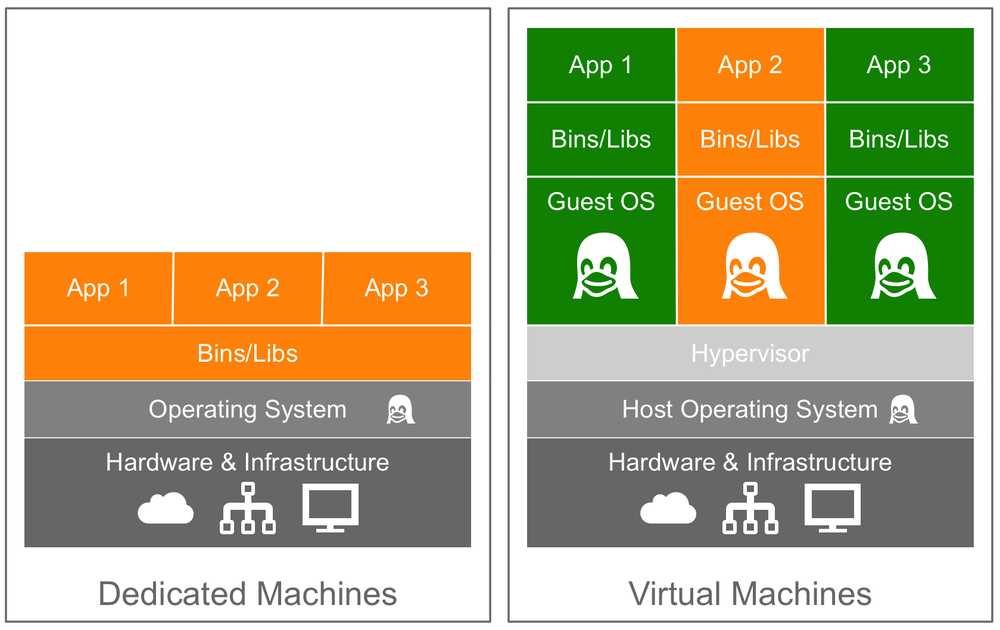
**BAB I**

**Dasar Teori**

**1.1 CONTAINER**

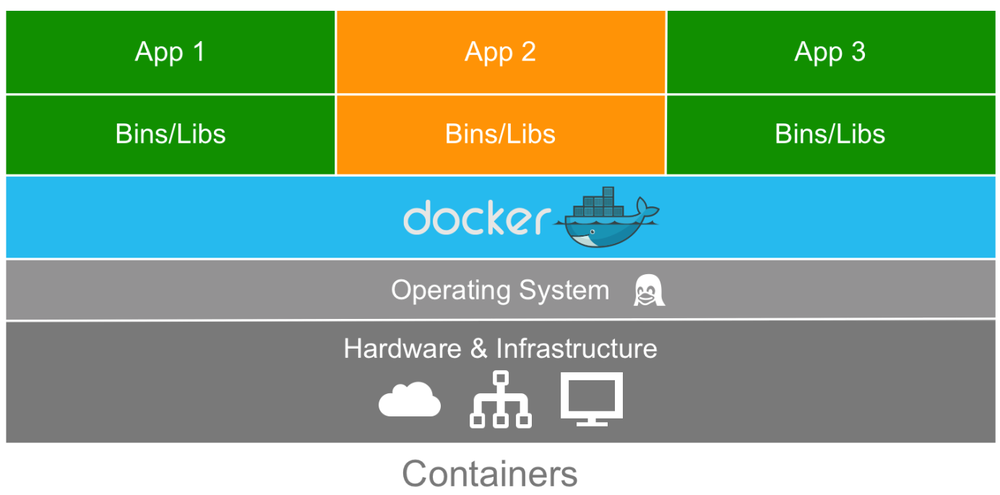
1.1.1 **Pengertian Container**

Pada pengembangan aplikasi secara konvensional, developer akan menyerahkan aplikasi yang telah selesai kepada bagian Operation untuk diinstall di server produksi, hal ini akan menjadi sulit jika aplikasi tersebut merasa bergantung pada beberapa kebutuhan seperti versi, library, dan depedensi yang lain. Untuk mengatasi tantangan yang ditimbulkan oleh beragam aplikasi dan kebutuhannya tersebut, maka salah satu solusinya adalah menggunakan Vitual Machine (VM) untuk mengisolasi aplikasi. Pada saat kita menjalankan VM, kita harus menginstall sistem operasinya terlebih dahulu. Misal kita membuat aplikasi yang lumayan banyak, mau tidak mau kita harus membuat VM yang lebih banyak juga. Dan tentu saja hal tersebut akan membutuhkan resource yang lebih banyak. Jadi pada saat kita ingin membuat VM lebih banyak, kita akan mengulangi proses installasi sampai VM itu sesuai dengan kebutuhan, dan tentu itu pekerjaan yang melelahkan dan memakan banyak waktu.



*sumber: ADINUSA (Akademi Digital Nusantara)*

Solusi lainnya adalah menggunakan Container. Berbeda dengan VM, Container sendiri berfokus pada sisi Aplikasi. Container sendiri sebenarnya berjalan diatas aplikasi Container Manager yang berjalan di sistem operasi, salah satu contoh conainer manager adalah Docker. Docker Container yang dapat meng-enkapsulasi ringan untuk aplikasi dan kebutuhannya tanpa harus menginstall sistem operasi tersendiri. Container memungkinkan developer untuk mengemas aplikasi, aplikasi dan dependency lainnya ke dalam unit yang terisolasi, sehingga memudahkan penerapannya di berbagai lingkungan.



*sumber: ADINUSA (Akademi Digital Nusantara)*

Container akan menggunakan sistem operasi host dimana Container managernya berjalan. Oleh karena itu, Container akan lebih hemat resource dan lebih cepat, karena tidak butuh sistem operasi tersendiri. Container ini dapat meminimalkan penggunaan recourse pada perangkat lunak dengan lebih konsisten dan portable, yang artinya pengembang dapat "membangun, mengirim, dan menjalankannya di mana saja," sehingga mendukung siklus deploy atau rilis aplikasi menjadi lebih singkat.

**1.2 DOCKER**

**1.2.1 Pengertian Docker**

Docker merupakan salah satu platform Container Manager yang populer. Docker memungkinkan developer untuk mengemas, mengirim, dan menjalankan aplikasi beserta dependensinya dalam lingkungan yang terisolasi yang disebut sebagai "container.". Sebagai container manager, Docker menyediakan berbagai fitur dan fungsionalitas yang memungkinkan pengguna untuk membuat, mengelola, dan menjalankan kontainer dengan mudah. Container Docker dapat berisi semua yang diperlukan oleh suatu aplikasi untuk berjalan, termasuk kode aplikasi, runtime, library, alat sistem, dan variabel lingkungan. Docker menggunakan teknologi kontainerisasi untuk memastikan bahwa aplikasi dapat berjalan dengan konsisten di berbagai lingkungan, baik itu di lingkungan pengembangan, pengujian, maupun produksi. Docker menyederhanakan proses pengembangan dan distribusi aplikasi dengan memastikan bahwa setiap kontainer berjalan secara independen dan dapat dipindahkan dari satu lingkungan ke lingkungan lain tanpa perlu memodifikasi kode atau konfigurasi aplikasi.

Fitur utama dari Docker sebagai container manager meliputi:

1. Pembuatan Kontainer: Docker memungkinkan pengguna untuk membuat kontainer dengan mudah dari image yang telah ditentukan sebelumnya atau melalui file Dockerfile yang berisi instruksi untuk membangun image.

2. Penyimpanan dan Penyebaran Image: Docker menyediakan repository pusat yang disebut Docker Hub dimana pengguna dapat menyimpan dan berbagi image Docker. Selain itu, Docker juga memungkinkan pengguna untuk membuat dan menyebarkan image Docker sendiri.

3. Pengaturan Jaringan: Docker memungkinkan pengguna untuk mengatur jaringan antar-kontainer atau antara kontainer dan host, termasuk konfigurasi seperti port forwarding dan pengaturan jaringan yang lebih kompleks.

4. Manajemen Resource: Docker memberikan kontrol yang baik atas sumber daya yang digunakan oleh kontainer, seperti CPU, memori, dan ruang disk.

5. Monitoring dan Logging: Docker menyediakan alat untuk memantau kesehatan dan kinerja kontainer, serta untuk mengumpulkan dan menganalisis log dari kontainer.

6. Otomatisasi dan Orkestrasi: Docker dapat diintegrasikan dengan alat-alat orkestrasi seperti Docker Swarm atau Kubernetes untuk otomatisasi pengelolaan kontainer dalam lingkungan yang lebih besar dan kompleks.

Dengan fitur-fitur ini, Docker memungkinkan pengguna untuk dengan mudah membuat, mengelola, dan menjalankan kontainer dengan efisien, menjadikannya pilihan utama sebagai container manager dalam pengembangan dan pengelolaan aplikasi modern.

**1.2.2 Docker Registry**

Docker Registry adalah repository tempat penyimpanan untuk Docker images. Ini berfungsi sebagai tempat sentral untuk menyimpan dan mengelola images Docker. Registry dapat bersifat publik atau privat, tergantung pada kebutuhan pengguna. Beberapa contoh Docker Registry yang populer termasuk Docker Hub (publik) dan AWS Elastic Container Registry (ECR) (privat). Pada Docker Hub sudah tersedia banyak images dari stack-stack terkenal seperti php, mysql, nginx, python, golang, dll.

Sistem Registry di Docker hampir sama dengan sistem git, dimana di Registry kita bisa pull (download image orang lain / dari suatu sumber) atau kita juga bisa push (membuat image kita sendiri lalu menguploadnya).

**1.2.3 Docker Images**

Pada Virtual Machine, images hampir sama dengan “Sistem Operasi” yang akan diinstall. Perbedaanya, Docker Images merupakan sebuah hasil building/ installer aplikasinya yang berisi aplikasi siap digunakan. Artinya apa? Image di docker ini adalah suatu keadaan dimana aplikasi sudah siap dijalankan, jadi bukan seperti Installer Sistem Operasi seperti di Virtual Machine. Seandainya kita memerlukan aplikasi berbasis linux, kita bisa mendownload image tersebut pada Docker Registry yang tersedia, contohnya yaitu Docker Hub.

**1.2.4 Docker Container**

Docker Container adalah instansi yang berjalan dari Docker Image. Docker Container adalah unit terisolasi yang dapat berkomunikasi dengan sistem operasi host, tetapi tetap terpisah dari container lainnya. Mereka menyediakan lingkungan yang konsisten dan dapat diulang, memungkinkan pengembang untuk menjalankan aplikasi tanpa khawatir tentang dependensi atau konfigurasi sistem operasi host. Container dapat dijalankan, dihentikan, dihapus, dan didistribusikan dengan cepat dan mudah.

**1.2.5 Dockerfile**

Dockerfile merupakan file teks biasa yang berisi serangkaian instruksi yang bisa kita gunakan untuk membuat sebuah Docker Image. Anggap saja semua instruksi untuk menjalankan aplikasi kita, kita simpan di dalam Dockerfile, nanti Dockerfile tersebut akan dieksekusi sebagai perintah untuk membuat Docker Image. Untuk membuat Docker Image dari Dockerfile, kita bisa menggunakan perintah **docker build**.

Ada banyak tipe instruksi yang bisa digunakan dalam Dockerfile, untuk lebih jelasnya kita bisa akses [di sini](https://docs.docker.com/reference/dockerfile/#overview). Tapi di sini kita hanya membahas beberapa saja yang penting, seperti berikut:

| **INSTRUKSI** | **DESKRIPSI** |
| --- | --- |
| **FROM** | Instruksi FROM memberi tahu Docker basis image apa yang akan digunakan. |
| **RUN** | RUN sebuah instruksi untuk mengeksekusi perintah di dalam image pada saat build stage. Biasanya digunakan untuk menginstal software dan menjalankan script, command, dan tugas-tugas lainnya. |
| **CMD** | CMD atau Command, merupakan instruksi yang digunakan ketika Docker Container berjalan |
| **ADD/COPY** | ADD dan COPY adalah instruksi yang dapat digunakan untuk menambahkan file dari source ke dalam folder destination di Docker Image |
| **USER** | USER adalah instruksi yang digunakan untuk mengubah user atau user group ketika Docker Image dijalankan |
| **WORKDIR** | WORKDIR adalah instruksi untuk menentukan direktori / folder untuk menjalankan instruksi RUN, CMD, ENTRYPOINT, COPY dan ADD |
| **LABEL** | Instruksi LABEL merupakan instruksi yang digunakan untuk menambahkan metadata ke dalam Docker Image yang kita buat |

**BAB II**

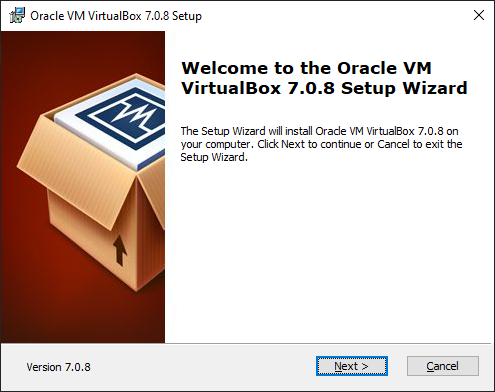
**Persiapan LAB**

**Minimum spesifikasi**

1. Windows 10 64bit
2. RAM 4GB
3. Storage tersisa 10GB
4. VirtualBox 7.0.0+

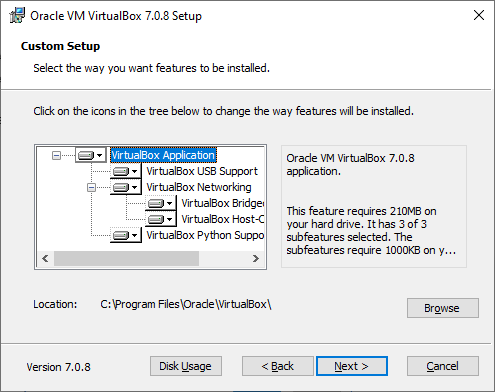
**Install VirtualBox**

1. Jalankan file installer VirtualBox



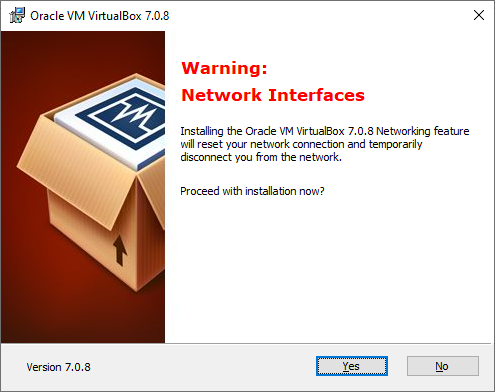
Klik next

1. Selanjutnya akan muncul menu fitur apa yang akan diinstall

****

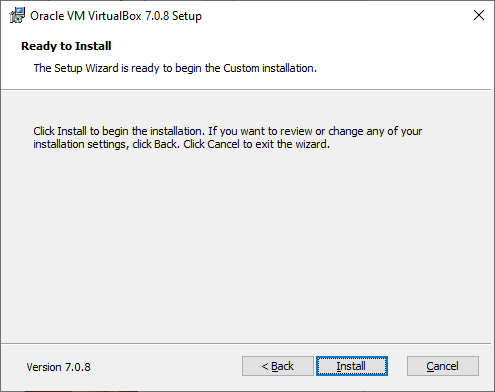
Untuk melanjutkan, klik next

1. Kemudian akan muncul menu untuk menginstall network interface



Klik “Yes”

1. Klik install untuk memulai proses instalasi

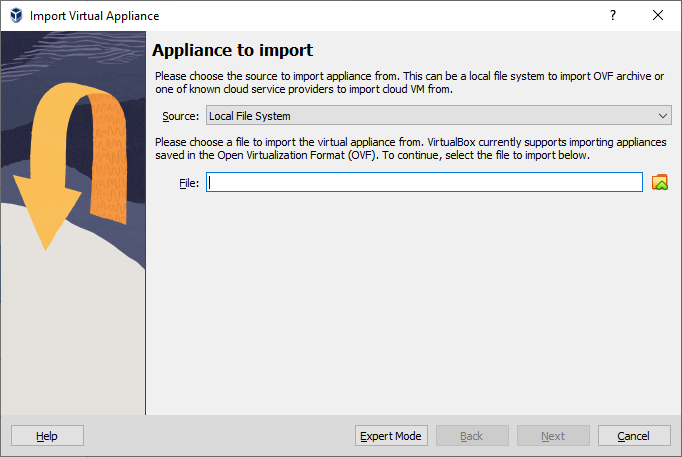


1. Instalasi selesai.

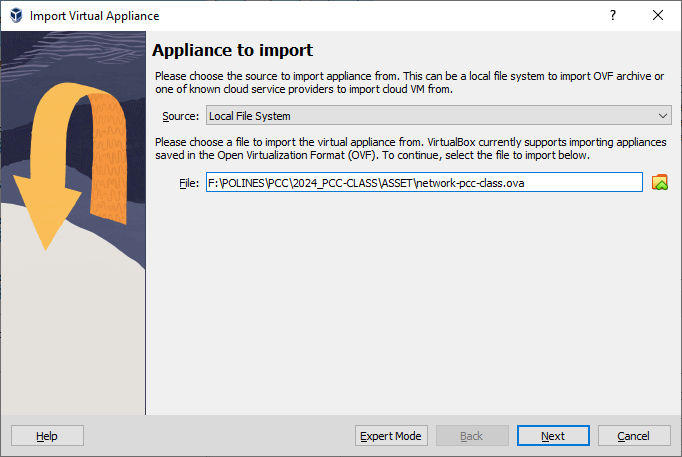


**Import Lab:**

1. Buka VirtualBox
2. Klik File > Import Appliance

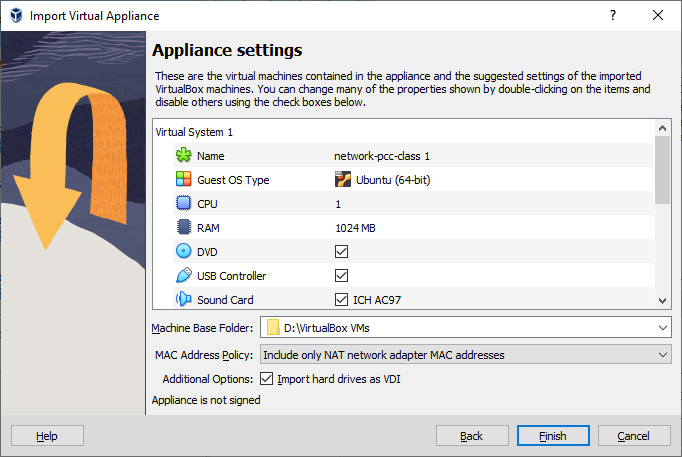


1. Kemudian pilih file .ova yang sudah di download sebelumnya.



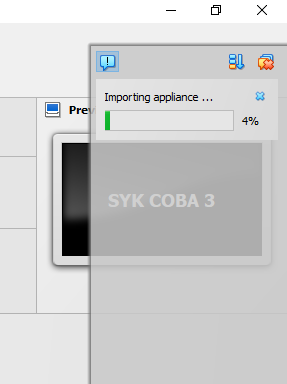
Klik Next

1. Kemudian akan muncul informasi dan pengaturan tentang VM yang akan di-import. Kita dapat mengubah beberapa opsi pengaturan tersebut, contohnya kita dapat mengubah nilai alokasi ram untuk Virtual M
2. achine kita.



Jika dirasa tidak ada yang ingin diubah, bisa langsung klik Finish.

1. Kemudian akan muncul notifikasi yang berisi progress proses di sebelah kanan window.



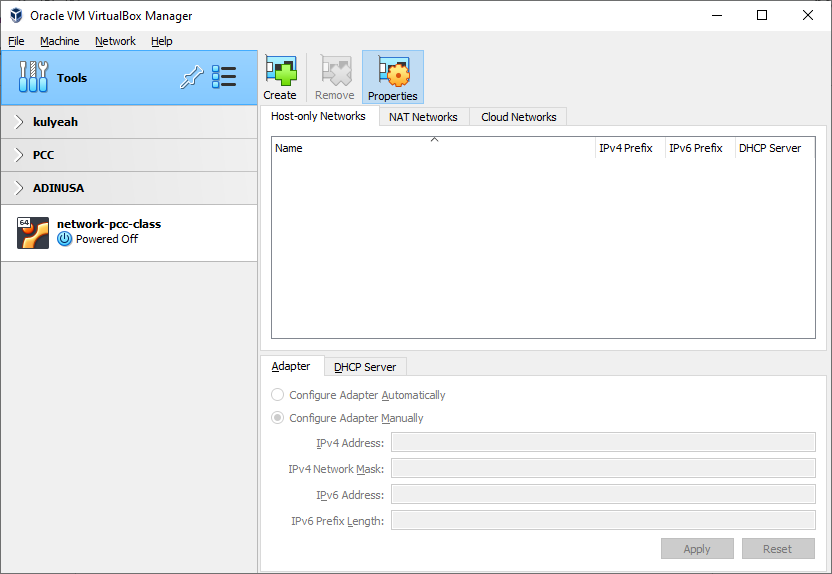
Tunggu hingga proses import selesai.

1. Proses import selesai.

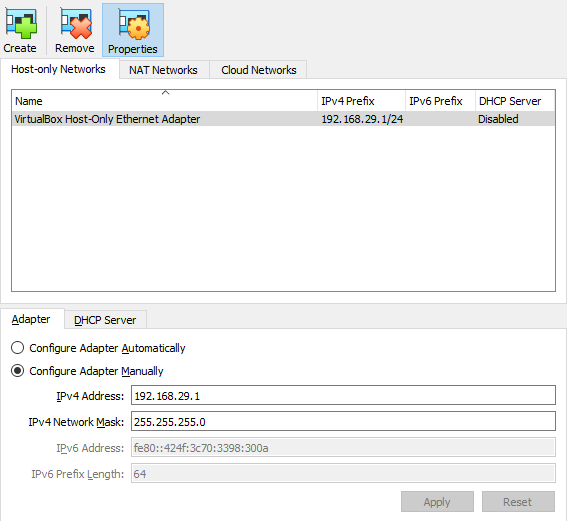
**Setup Network**

Disini kita akan mengatur network interface yang nantinya akan kita gunakan untuk akses internet dan local VM Ubuntu kita.

1. Masuk ke menu ‘Tools’, kemudian klik tab ‘Host-only Network’

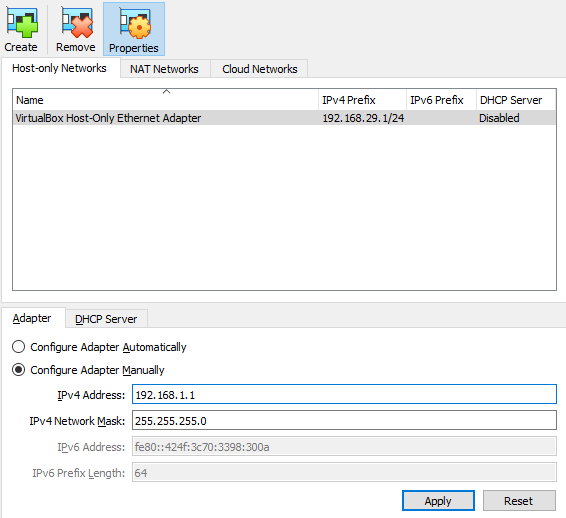


1. Klik Create. Kemudian tunggu sampai proses membuat network interface selesai.
2. Jika pembuatan network interface sudah selesai, maka akan muncul Host-only interface baru beserta dengan IPnya



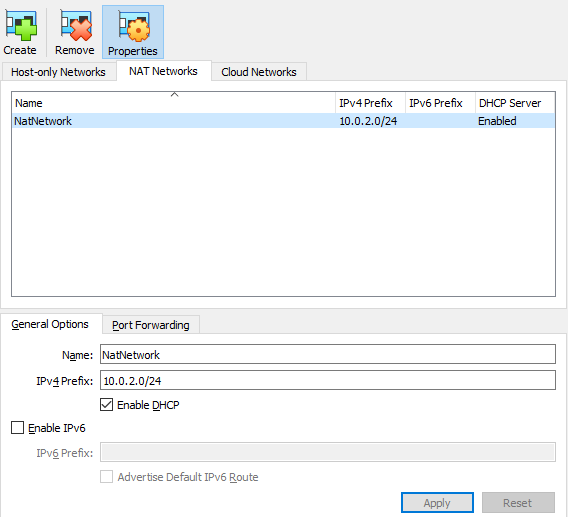
1. Lalu isi dengan IP sebagai berikut:

192.168.1.1



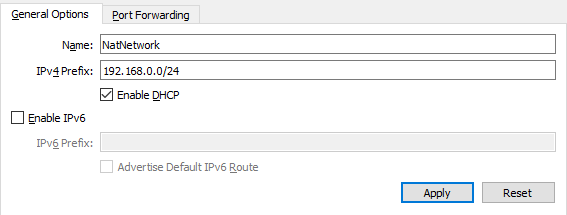
Kemudian klik ‘apply’

1. Pindah ke tab ‘NAT Network’, kemudian klik ‘Create’ untuk membuat interface NAT baru

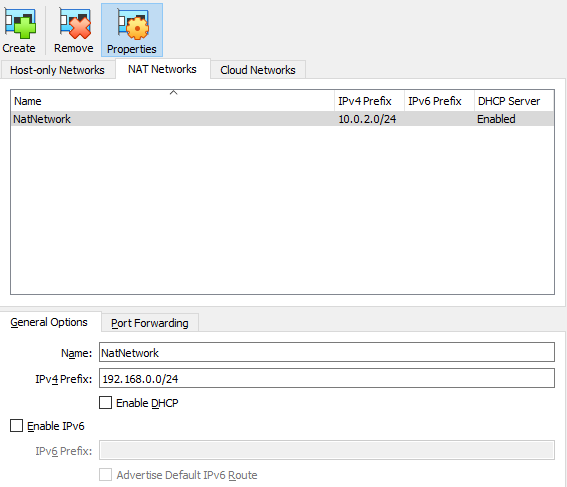


1. Ubah IP sebagai berikut:

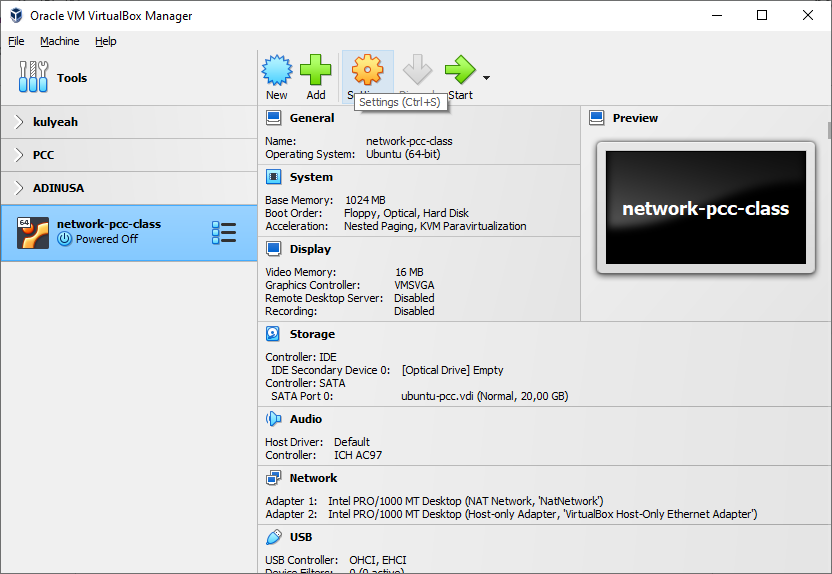
192.168.0.0/24



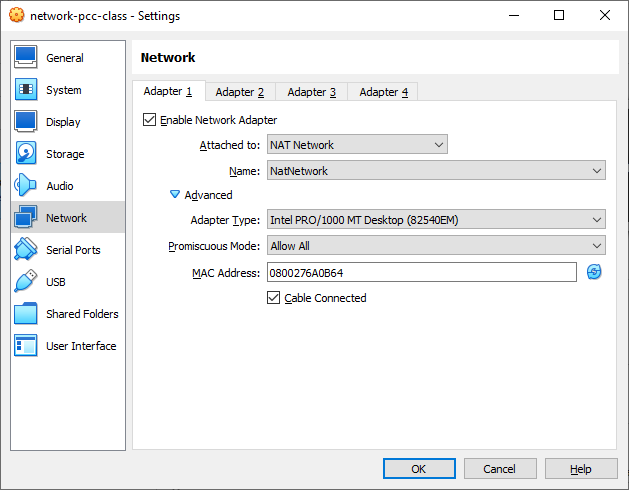
1. Uncheck Enable DHCP. Kemudian klik apply



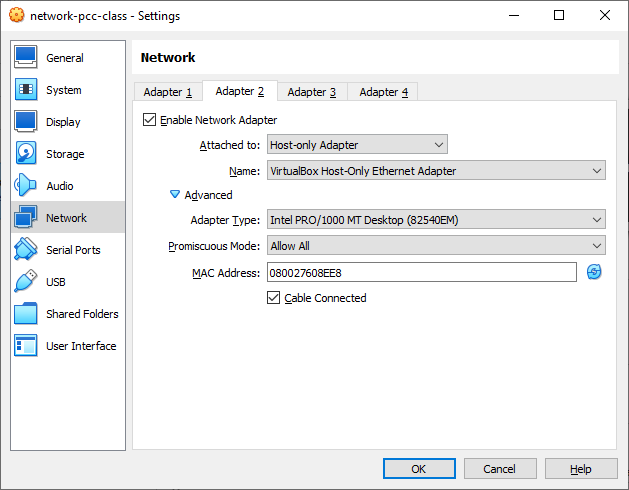
1. Klik VM, kemudian masuk ke settings



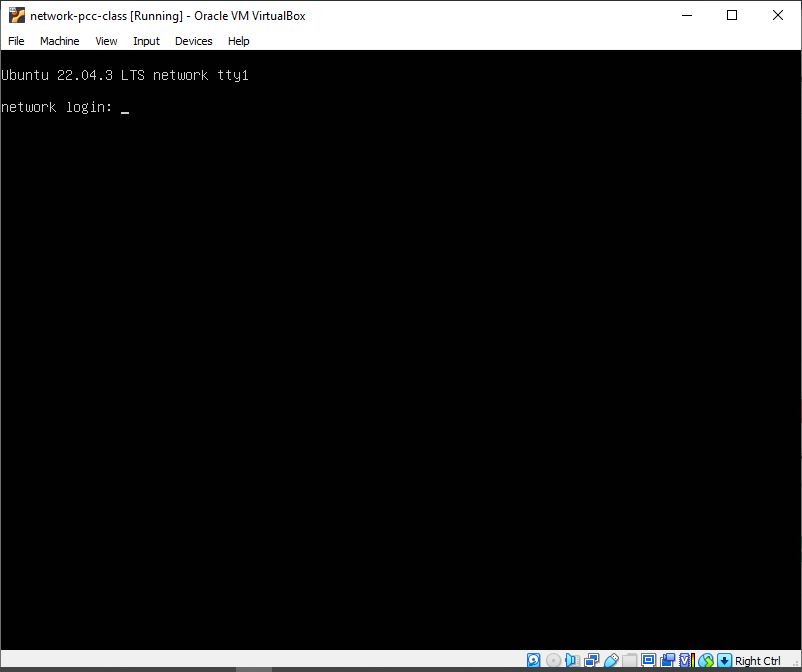
1. Masuk ke tab ‘Network’.
2. Pada ‘Adapter 1’, pilih Nat Network.



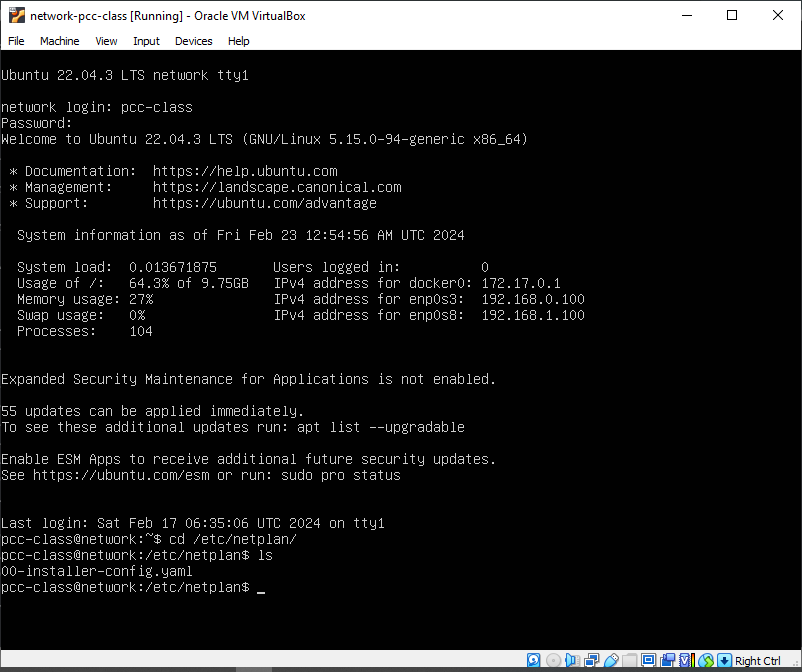
1. Lalu pada ‘Adapter 2’, pilih Host-only Adapter



1. Selanjutnya kita akan konfigurasi IP untuk linuxnya.
2. Hidupkan VM

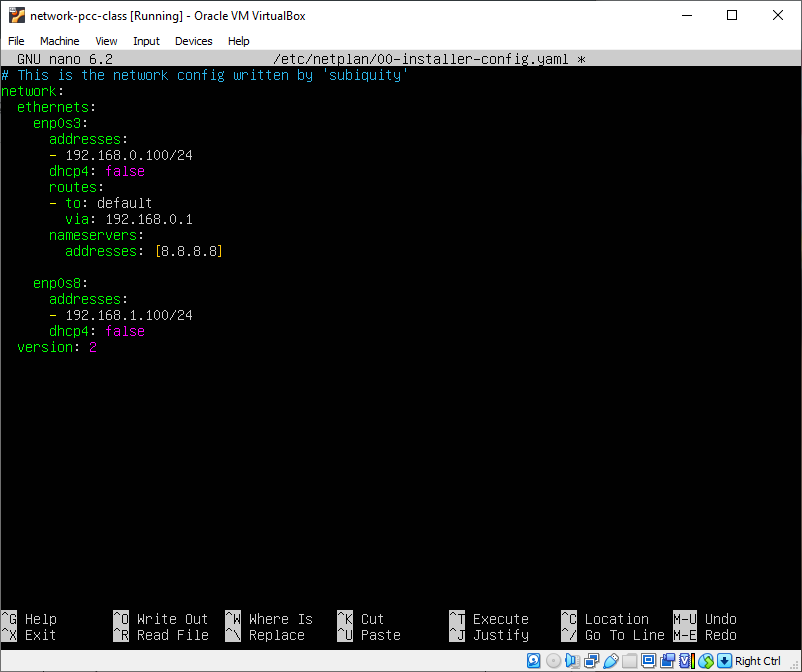


1. Login dengan username: **pcc-class** dan password: **network2024**
2. Kemudian masuk ke direktori netplan dimana kita akan mengkonfigurasi network Ubuntu disana. Ketikkan perintah **cd /etc/netplan**

****

Direktori **/etc/netplan** digunakan untuk mengkonfigurasi jaringan pada VM Ubuntu versi 22 yang kita gunakan sekarang ini.

1. Kita edit file 00-installer-config.yaml dengan mengetikkan perintah **sudo nano 00-installer-config.yaml**
2. Kemudian edit seperti berikut:

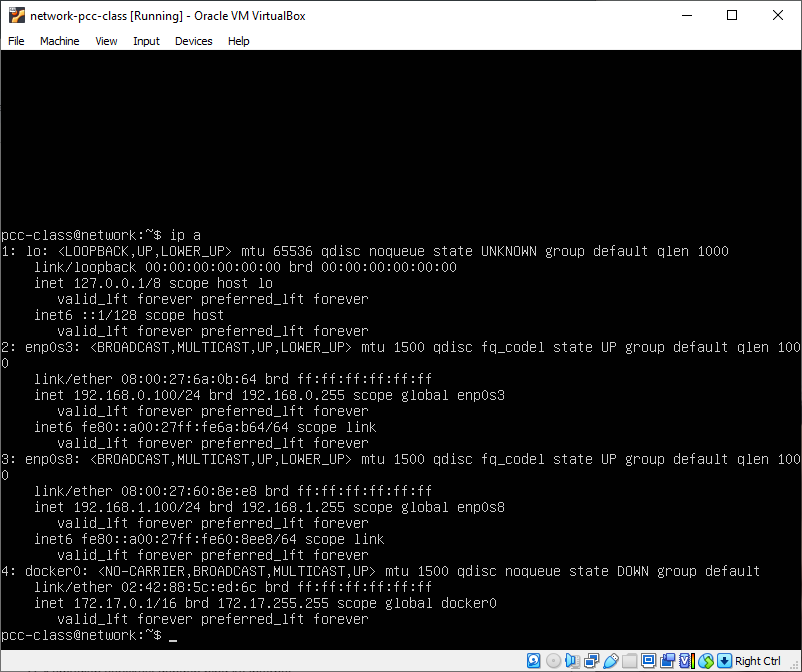


| **network:**  **ethernets:**  **enp0s3:**  **addresses:**  **- 192.168.0.100/24**  **dhcp4: false**  **routes:**  **- to: default**  **via: 192.168.0.1**  **nameservers:**  **addresses: [8.8.8.8]**  **enp0s8:**  **addresses:**  **- 192.168.1.100/24**  **dhcp4: false**  **version: 2** |
| --- |

Dapat dilihat, kita akan mengkonfigurasi 2 network interface, yakni **enp0s3** dan **enp0s8** dimana masing-masing interface menghubungkan 2 adapter NAT dan Host-only sesuai dengan adapter yang kita atur untuk VM Ubuntu kita.

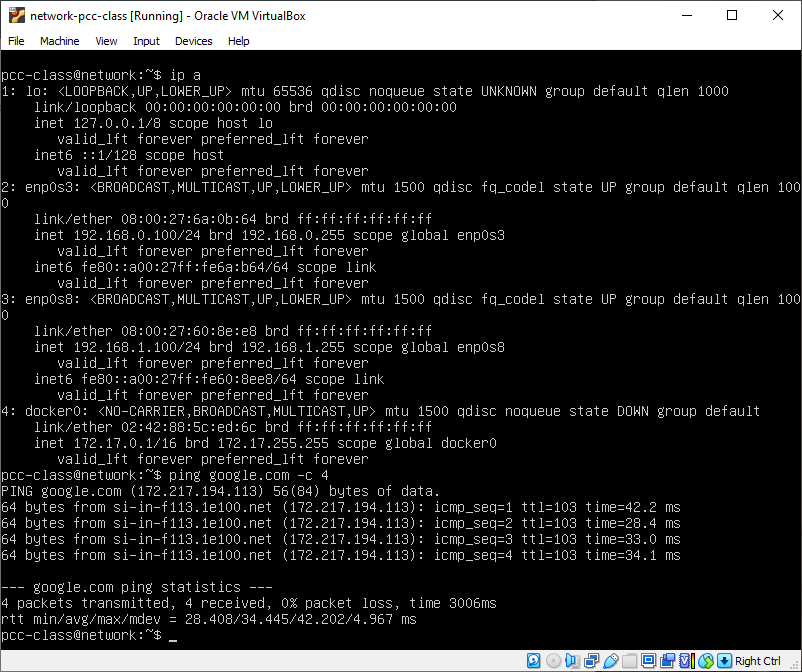
* **enp0s3** → Adapter 1 (NAT Network)
* **enp0s8** → Adapter 2 (Host-only Network)

1. Simpan dengan mengetikkan **ctrl+X**, **Y**, **Enter**
2. Gunakan perintah **sudo netplan apply** untuk menerapkan konfigurasi jaringan terbaru
3. Lalu Verifikasi dengan perintah **ip a**.



1. Kemudian verifikasi dengan ping ke internet

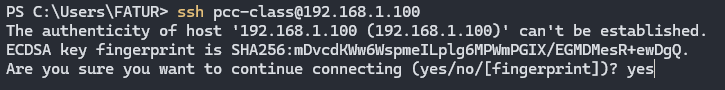
| **ping google.com** |
| --- |



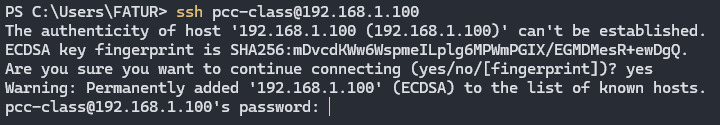
**Instalasi Tools**

Dalam praktek selanjutnya, kita akan banyak copy-paste script. Dikarenakan dalam Ubuntu CLI tidak bisa menggunakan copy-paste, maka kita akan menggunakan alternatif lain, yaitu remote SSH melalui Terminal atau Command Prompt Windows.

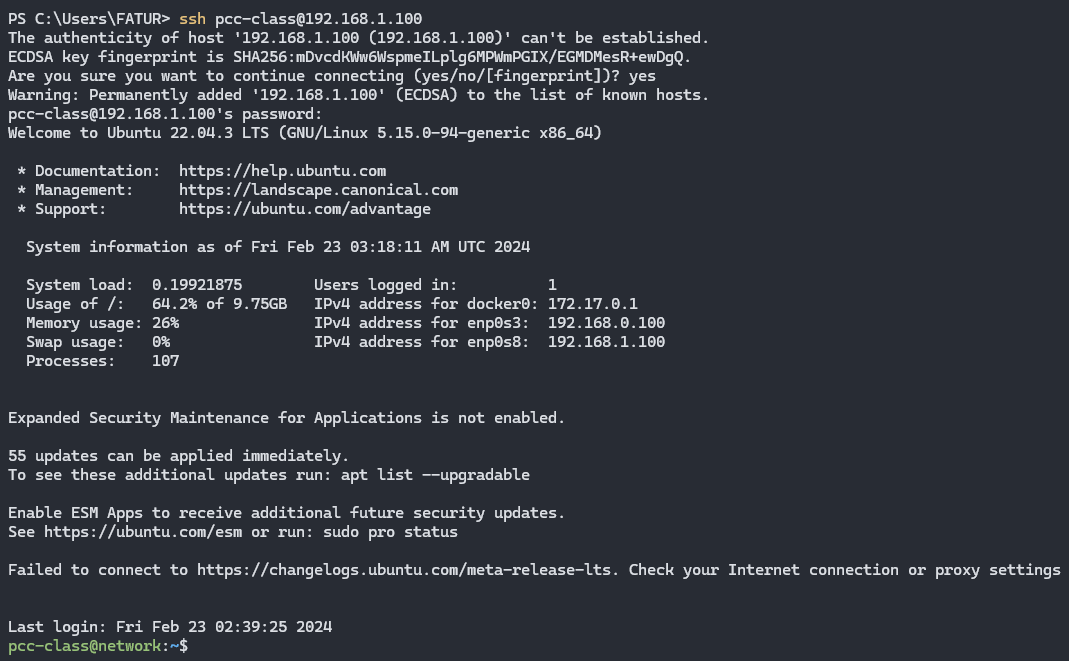
1. Buka aplikasi Terminal atau Command Prompt pada windows.
2. Kemudian ketikkan ssh pcc-class@192.168.1.100



Jika muncul dialog untuk meminta RSA-Key, ketik **yes**



1. Kemudian Masukkan password **network2024**

****

Login telah sukses

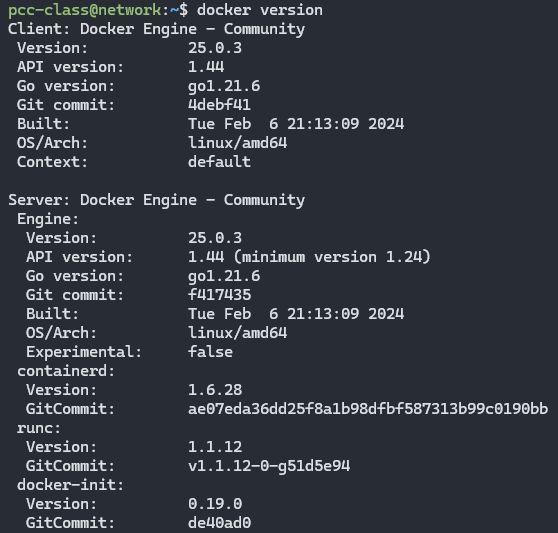
1. Selanjutnya kita akan meginstall docker. Jalankan perintah berikut:

| **sudo apt-get update**  **sudo apt-get install ca-certificates curl gnupg lsb-release -y**  **sudo mkdir -m 0755 -p /etc/apt/keyrings**  **curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg**  **echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \**  **$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**  **sudo apt-get update**  **sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y** |
| --- |

Paste dan jalankan kode tersebut pada Terminal

1. Jika instalasi telah selesai, verifikasi dengan menjalankan perintah

| **docker version** |
| --- |



1. Tambahkan user ke Docker Group

| **groupadd docker**  **sudo usermod -aG docker $USER**  **sudo chmod 666 /var/run/docker.sock** |
| --- |

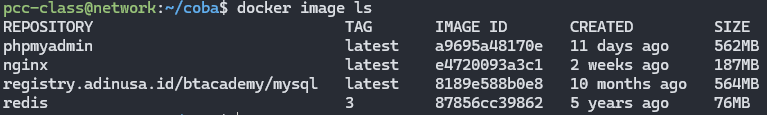
**BAB III**

**Langkah-langkah**

**Manajemen Docker Image**

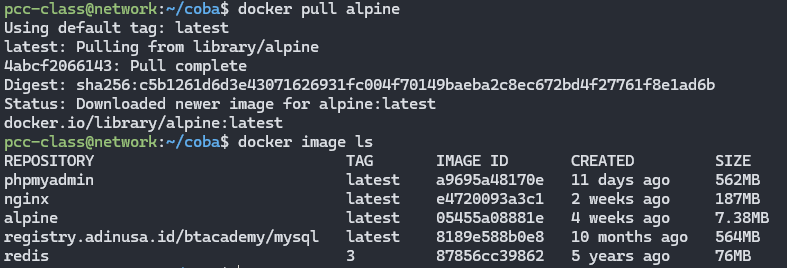
1. Untuk melihat image yang ada, jalankan perintah

**docker image ls**



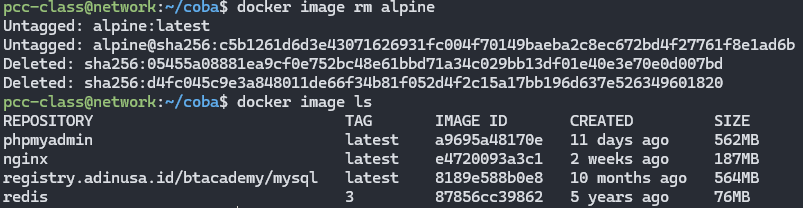
1. Untuk mengambil image dari registry, dapat gunakan perintah

**docker pull <nama-image>**



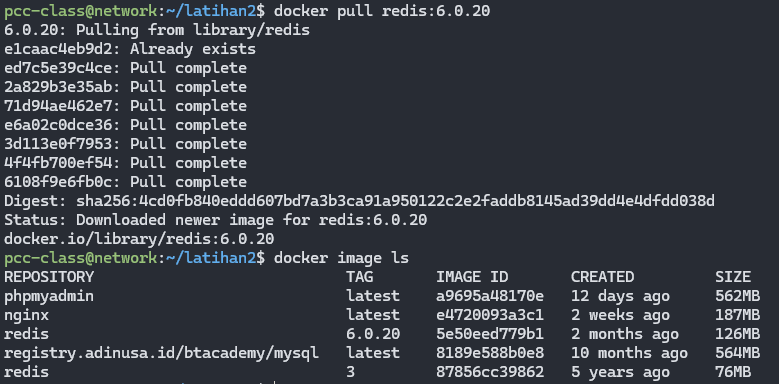
1. Untuk menghapus image, gunakan perintah

**docker image rm <nama/image-id>**



1. Untuk mengambil image dengan versi/tag tertentu, gunakan perintah

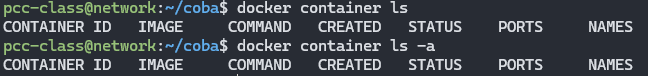
**docker image pull <nama-image:tag>**



**Manajemen container**

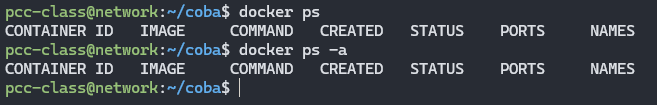
1. Untuk melihat container yang ada, gunakan perintah

**docker container ls**



Karena kita belum menjalankan container, maka pada saat kita list container, outputnya masih kosong.

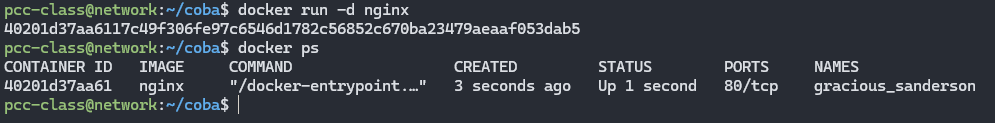
Kita juga dapat menjalankan perintah **docker ps** untuk melihat container yang ada



1. Untuk menjalankan container, gunakan perintah

**docker run -d <nama-image>**

contoh: **docker run -d nginx**

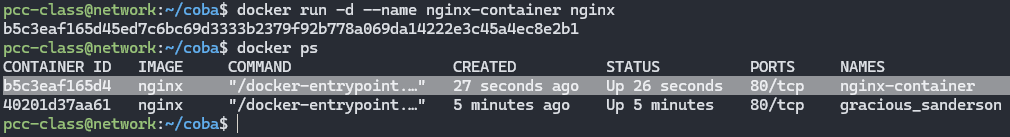


Container dari image nginx telah berjalan. Dapat dilihat disana terdapat container yang berjalan dari image nginx. Nama dari container akan otomatis diberikan oleh docker jika kita tidak memberi nama pada saat membuat container.

1. Untuk memberi nama container, gunakan perintah

**docker run -d --name <nama-container> <image>**

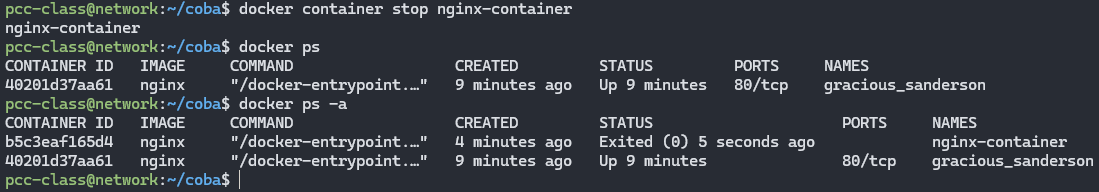
contoh: **docker run -d --name nginx-container nginx**



1. Menghentikan container yang sedang berjalan

**docker container stop <nama/id-container>**

contoh: **docker container stop nginx-container**

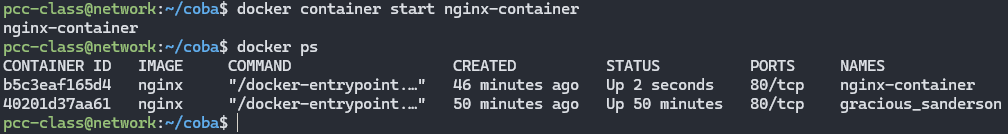


Dapat dilihat status pada container my-container berubah menjadi “Exited 5 seconds ago”.

1. Menghidupkan container

**docker container start <nama/id-container>**

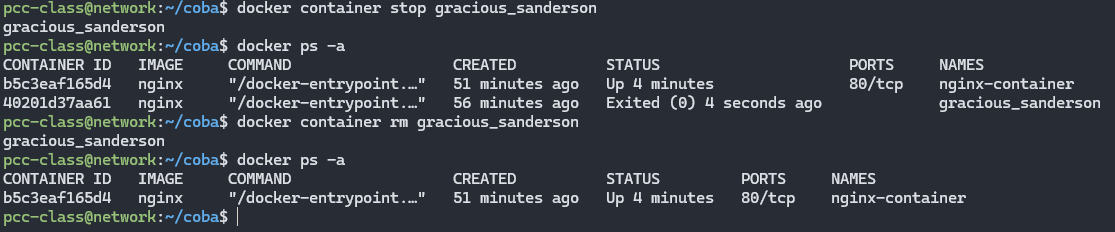
contoh: **docker container start nginx-container**



Dapat dilihat status container sudah berjalan

1. Menghapus container

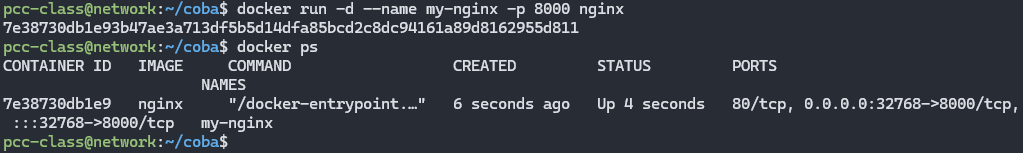
**docker container rm <nama/id-container>**



1. Kita juga dapat menjalankan docker di port tertentu

**docker run -d --name <nama-container> -p <port> <image>**

contoh: **docker run -d --name my-nginx -p 8000 nginx**

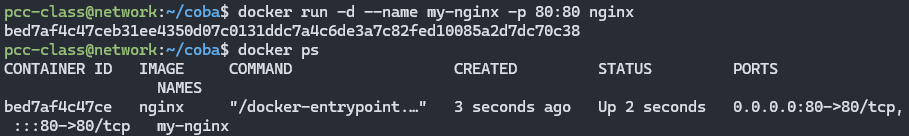


1. Kita dapat meng-expose aplikasi container kita

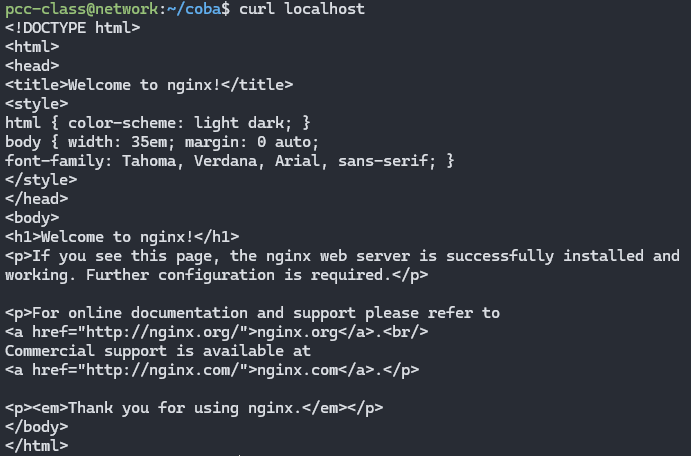
**docker run -d –name <nama-container> -p <port-keluar>:<port-conatiner> <image>**

contoh: **docker run -d --name my-nginx -p 80:80 nginx**

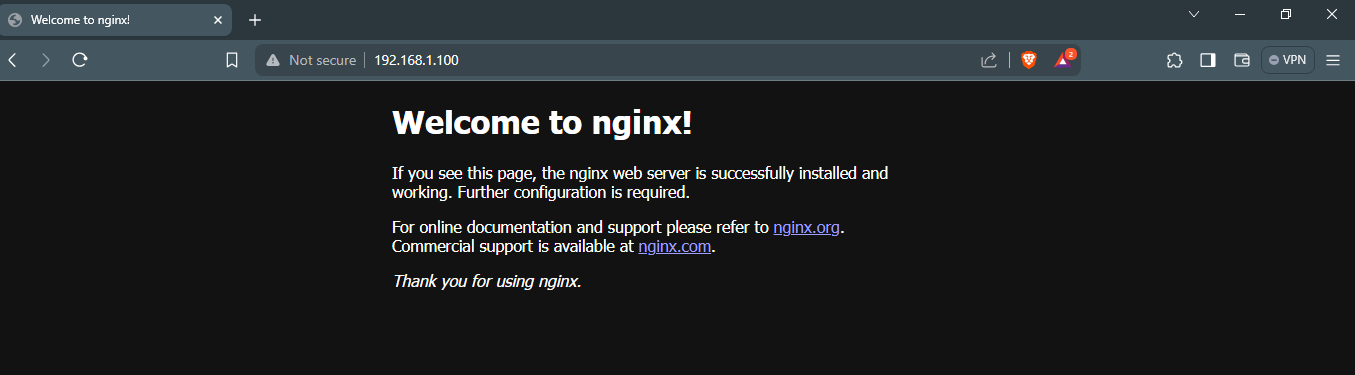
artinya semua perangkat pada jaringan kita akan dapat mengakses container nginx pada port 80.



Kita coba cek dengan perintah curl apakah service nginx berjalan



Kita coba cek juga menggunakan web browser pada laptop kita



1. Remote container

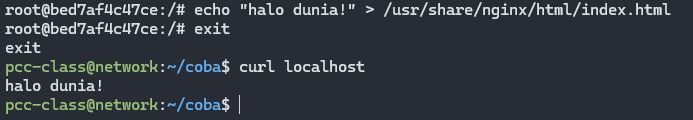
**docker exec -it <nama/id-container> /bin/bash**

contoh: **docker exec -it my-nginx /bin/bash**

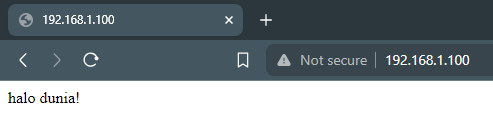


kita telah berhasil remote containernya

Kita coba untuk mengedit file tampilan nginx



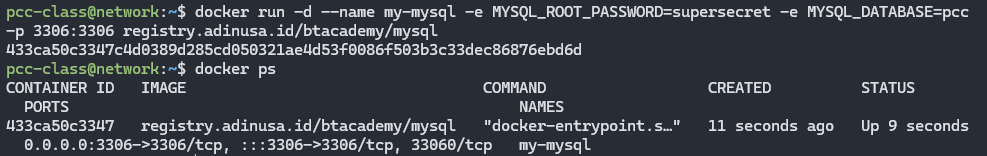
coba cek di web browser



1. Environment Variable

Untuk memasukkan Evironmnt Variable pada docker container kita, dapat menggunakan perintah **docker run -d -e <ENV=VALUE>**

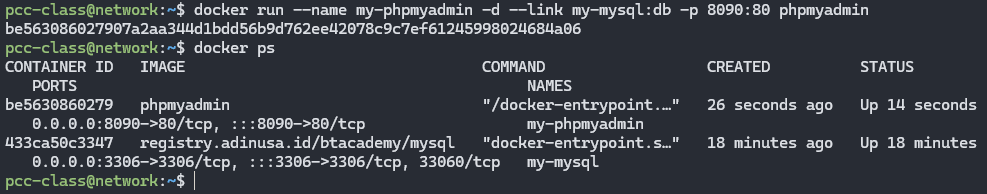
contoh: **docker run -d --name my-mysql -e MYSQL\_ROOT\_PASSWORD=supersecret -e MYSQL\_DATABASE=pcc -p 3306:3306**



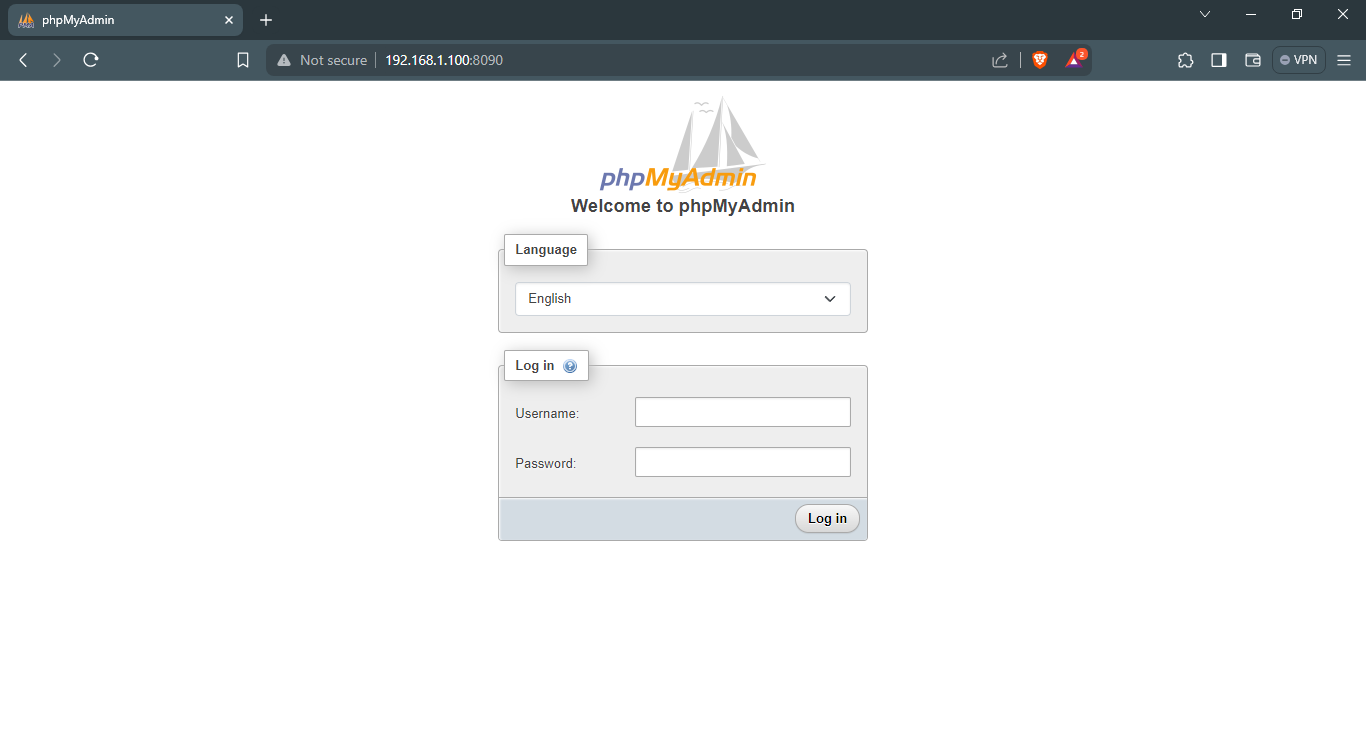
1. Membuat link dengan container lain

**docker run -d --name <name> --link <link ke container> <image>**

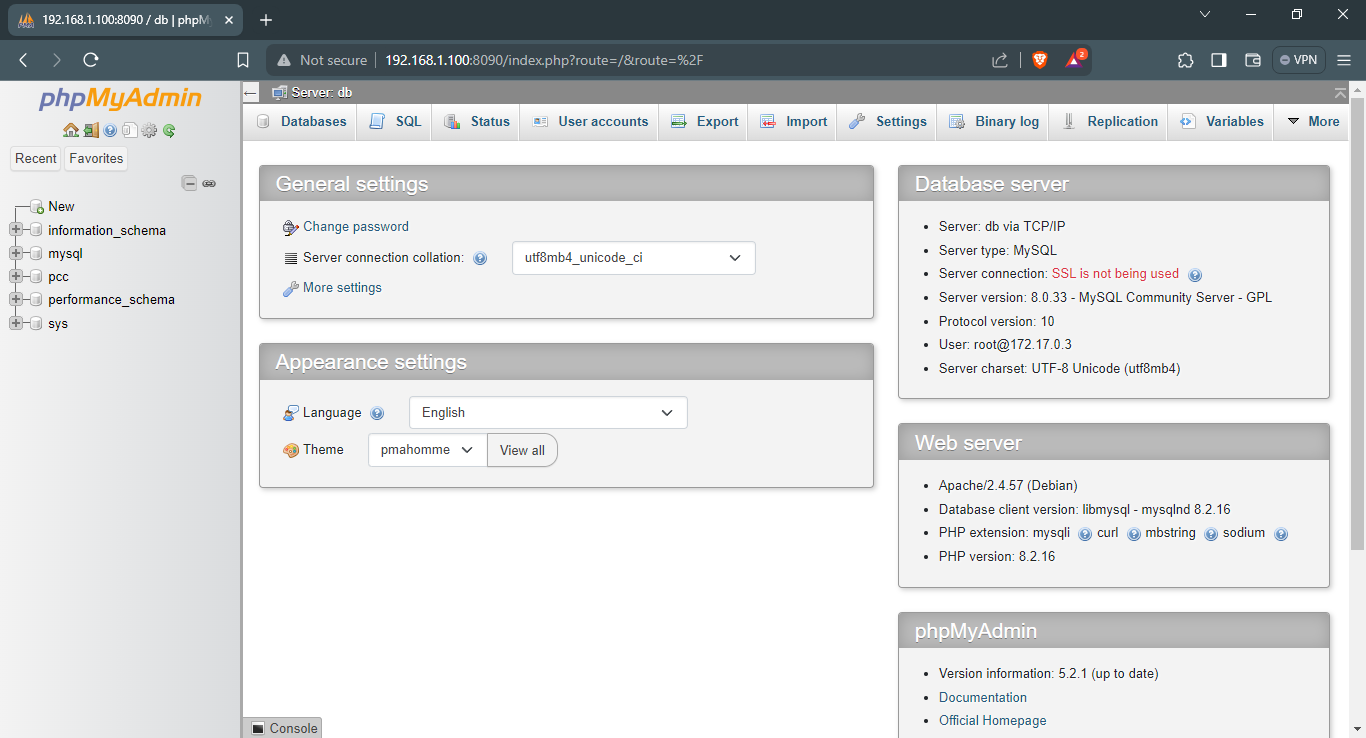
contoh: **docker run -d --name my-phpmyadmin --link my-mysql:db -p 8090:80 phpmyadmin**



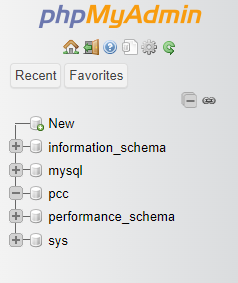
kita coba cek dengan mengakses lewat web browser dengan port 8090



login dengan username: **root**, password: **supersecret**



Dapat dilihat di sebelah kiri terdapat database bernama pcc



**Membuat docker image**

1. Buat sebuah file dengan nama Dockerfile (huruf D harus kapital)
2. Instruksi FROM

Mengambil image yang sudah ada

| **FROM alpine** |
| --- |

1. Instrusi RUN

Menjalankan perintah saat proses build

| **FROM alpine**  **RUN apk update && apk add --no-cache python3**  **RUN mkdir /coba** |
| --- |

Menjalankan perintah update repository apk alpine

1. Instruksi WORKDIR

| **FROM alpine**  **RUN apk update && apk add --no-cache python3**  **RUN mkdir /coba**  **WORKDIR /app** |
| --- |

Pada saat container dijalankan, maka akan langsung masuk ke direktori tujuan

1. Instruksi ADD/COPY

| **FROM alpine**  **RUN apk update && apk add --no-cache python3**  **RUN mkdir /coba**  **WORKDIR /app**  **ADD script.py /app/** |
| --- |

1. Instruksi CMD

Mengeksekusi command seteah fase build

| **FROM alpine**  **RUN apk update && apk add --no-cache python3**  **RUN mkdir /coba**  **WORKDIR /app**  **ADD script.py /app/**  **CMD ["python3", "script.py"]** |
| --- |

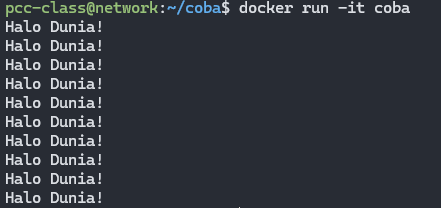
1. Buat sebuah file dengan nama script.py untuk uji coba

| **for i in range(10):**  **print("Halo Dunia!")** |
| --- |

1. Jika digabungkan akan menjadi seperti berikut:

| **# Gunakan image alpine terbaru yang berukuran kecil**  **FROM alpine:latest**  **# Jalankan perintah update dan install paket yang diperlukan**  **RUN apk update && apk add --no-cache python3**  **# Set folder kerja**  **WORKDIR /app**  **# Tambahkan file script.py ke dalam folder kerja**  **ADD script.py /app/**  **# Set perintah default saat container dijalankan**  **CMD ["python3", "script.py"]** |
| --- |

1. Simpan file
2. Build dengan perintah **docker build -t coba .**
3. Kemudian jalankan **docker run -it coba**

****

Hasil outputnya sama seperti script python yang sudah kita buat.