**LAPORAN PRAKTIKUM**

**TEKNOLOGI CLOUD**

**PERTEMUAN KE – 7**

****

**Disusun Oleh :**

**NAMA : TARISA DWI SEPTIA**

**NIM : 205410126**

**JURUSAN : TEKNIK INFORMATIKA**

**JENJANG : S1**

**Sekolah Tinggi Management Informatika Komputer**

**AKAKOM**

**YOGYAKARTA**

**2020**

Modul 7

Pengenalan Container Docker

1. **Tujuan**

* Dapat mengimplementasikan dan memahami container docker

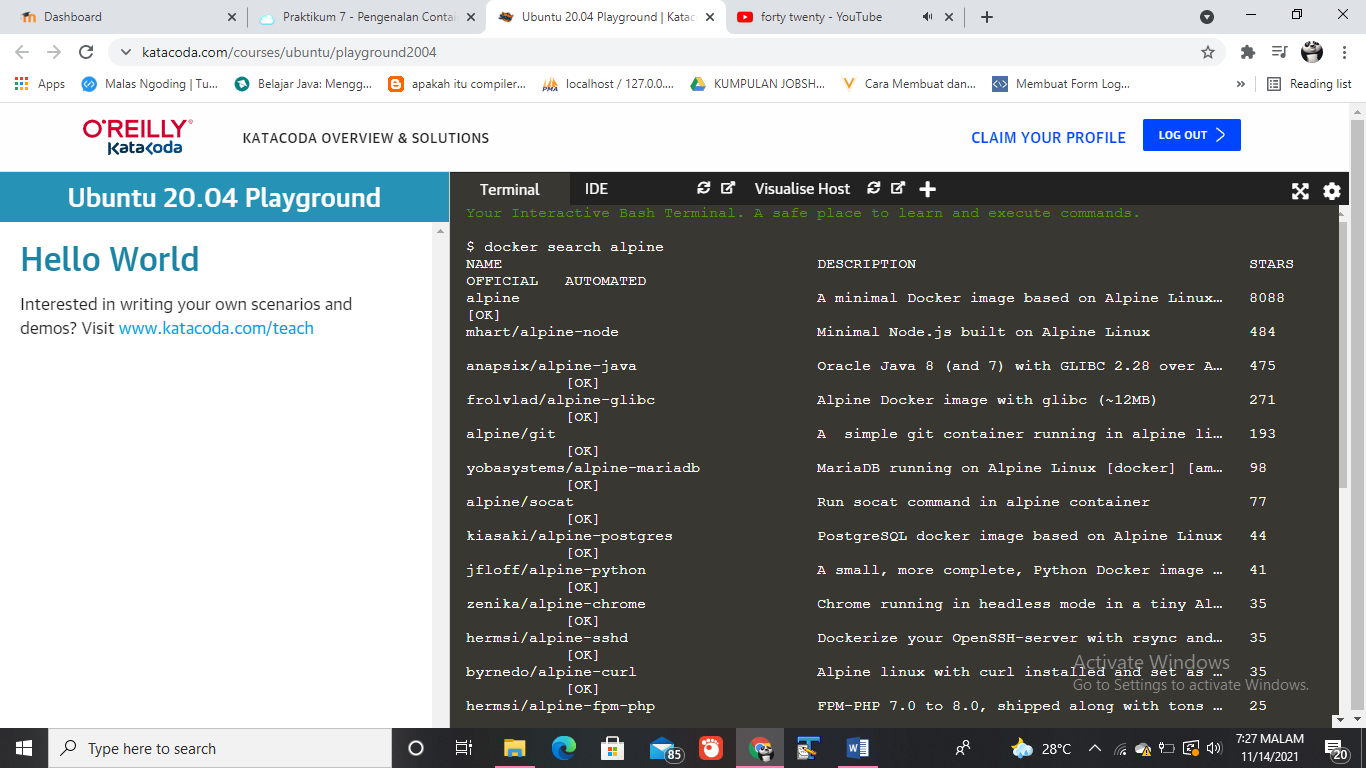
1. **Dasar Teori**

Dengan container, sebuah program ‘diikat’ beserta library-nya, file konfigurasi, dan seluruh hal yang dibutuhkannya. Perbedaan yang sangat terlihat dibandingkan dengan virtualisasi adalah container memiliki ukuran file yang jauh lebih kecil karena tidak perlu menyiapkan sistem operasi secara penuh. Dalam hal ini, pengembang biasa menyebutnya sebagai ‘lightweight’ platform. Aplikasi yang berjalan menggunakan container pun jauh lebih cepat dan lebih efisien. Docker adalah salah satu platform yang dibangun berdasarkan teknologi container. Docker merupakan sebuah project open-source yang menyediakan platform terbuka untuk developer maupun sysadmin untuk dapat membangun, mengemas, dan menjalankan aplikasi dimanapun sebagai sebuah wadah (container) yang ringan. Dengan sangat populernya docker, sebagian orang sering menganggap docker adalah sebutan lain untuk container.

1. **Praktik**
2. Menjalankan container

Semua container berawal dari docker image. Docker image merupakan sistem operasi yang didalamnya terdapat library / runtime / aplikasi tertentu. Image ini memiliki semua yang dibutuhkan untuk menjalankan proses sehingga host tidak perlu melakukan konfigurasi atau dependensi (ketergantungan terhadap packet atau library). Untuk menjalankan container, dapat menggunakan Docker image yang buat sendiri (Build) atau menggunakan Docker image yang telah disediakan oleh Docker dan komunitas. Beberapa perintah docker memerlukan penggunaan sudo, namun pada docker for windows dan katacoda, anda tidak perlu menggunakan perintah sudo. Dapat menggunkan perintah :

docker search <nama-image>

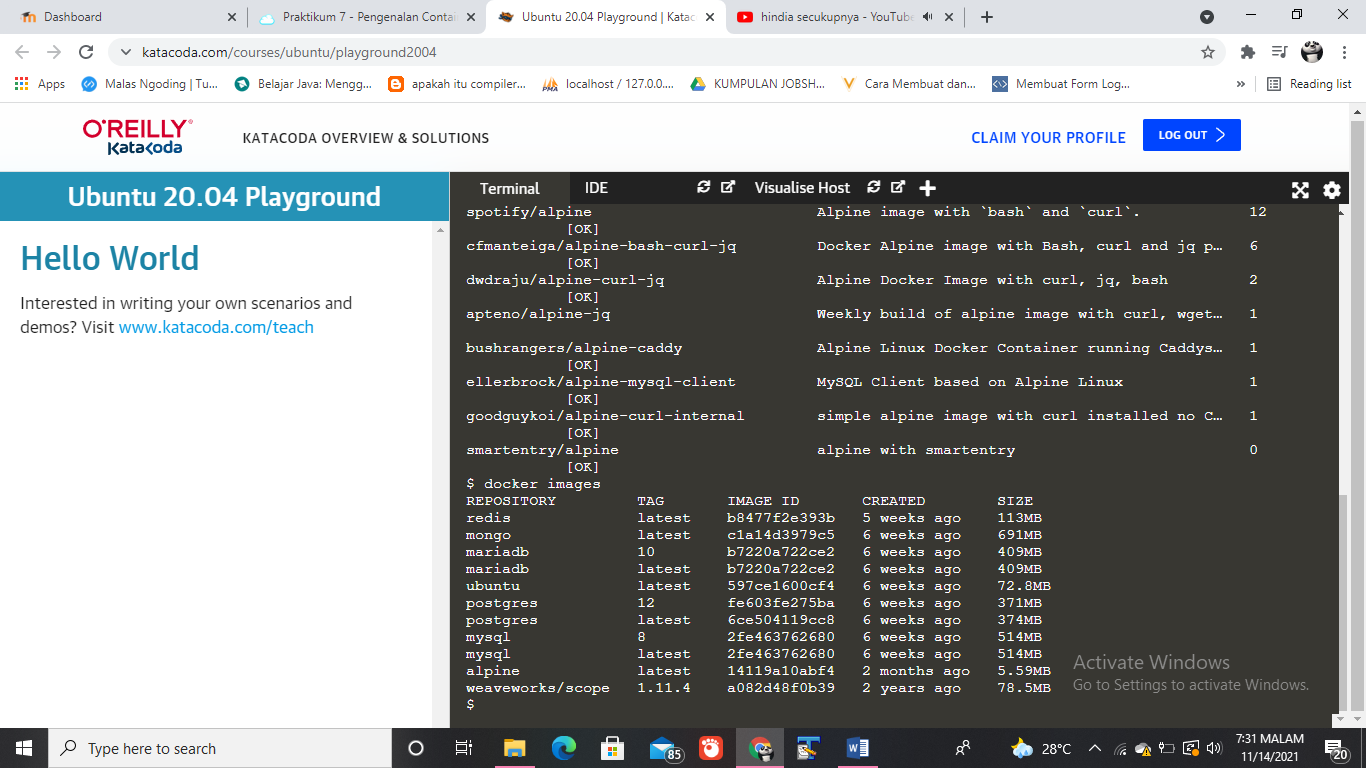
Output yang dihasilkan :

Secara detail anda dapat melihat nama image, deskripsi singkat, stars, official dan automated. Stars menunjukkan popularitas dari image tersebut, sementara official adalah versi stable yang dikeluarkan langsung oleh pembuat / pengembang aplikasi / sistem operasi. Direkomendasikan untuk memilih Docker image berdasarkan kedua kriteria ini.

Selanjutnya, jalankan container secara background menggunakan image yang tersedia atau yang telah anda pilih setelah proses searching image. Gunakan perintah **docker run <options> <image-name>** untuk menjalankan container. Secara default, Docker akan menjalankan perintah menggunakan foreground mode, sehingga untuk menjalankan di mode backgound, anda perlu menambahkan opsi -d. Pada langkah ini, anda tidak memiliki Docker image di komputer anda, sehingga akan mendownload langsung dari Docker registry. Jika anda menjalankan container untuk kedua kalinya, maka local image yang akan digunakan.

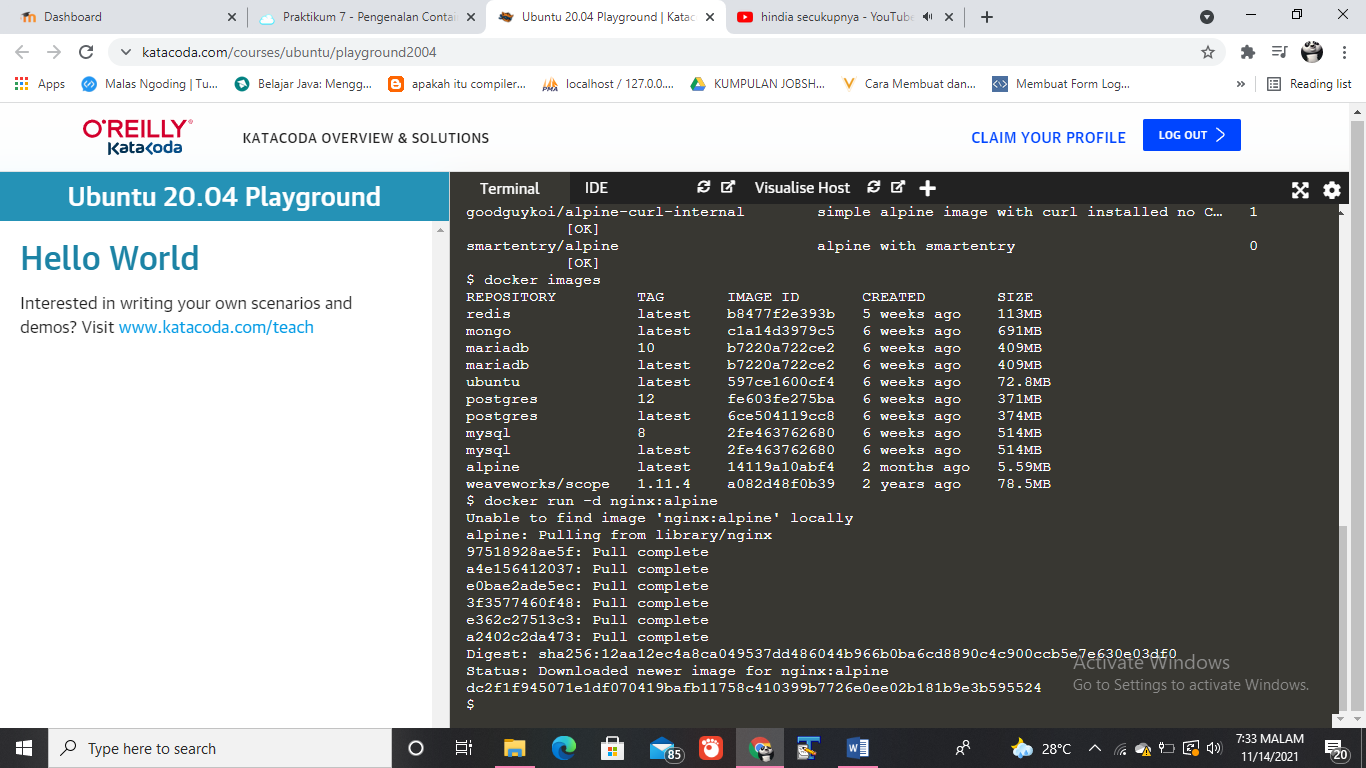
Untuk melihat Docker images di host, anda dapat gunakan perintah :

docker images

Hasil output :

Sekarang kita bisa mencoba untuk menjalankan container. Agar mudah diingat, anda dapat menggunakan opsi --name sehingga container tersebut mudah untuk dikenali. gunakan perintah :

docker run -d nginx:alpine

Hasil output :

Dalam proses ini akan menjalankan image alpine sebagai container, karena Docker image alpine belum terdapat di local.

1. Melihat proses docker container

Docker ps dapat digunakan untuk melihat listing proses container yang berjalan, gunakan perintah :

docker ps

Hasil output :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAME |
| dc2f1f945071 | nginx:alpine | "/docker-entrypoint.…" | 3 minutes ago | Up 3 minutes | 80/tc | xenodochial\_  kapitsa |

Untuk melihat lebih detail terkait volume, network, dan detail lain dari Docker container yang telah berjalan, anda dapat gunakan perintah :

docker inspect <container id / friendly name>

Sekarang kita coba lihat kembali hasil dari perintah sebelumnya tadi dengan menggunakan docker ps. pada kolom tersebut terdapat container ID. Ambil 3 karakter didepannya untuk menguji:

docker inspect dc2

Hasil output :

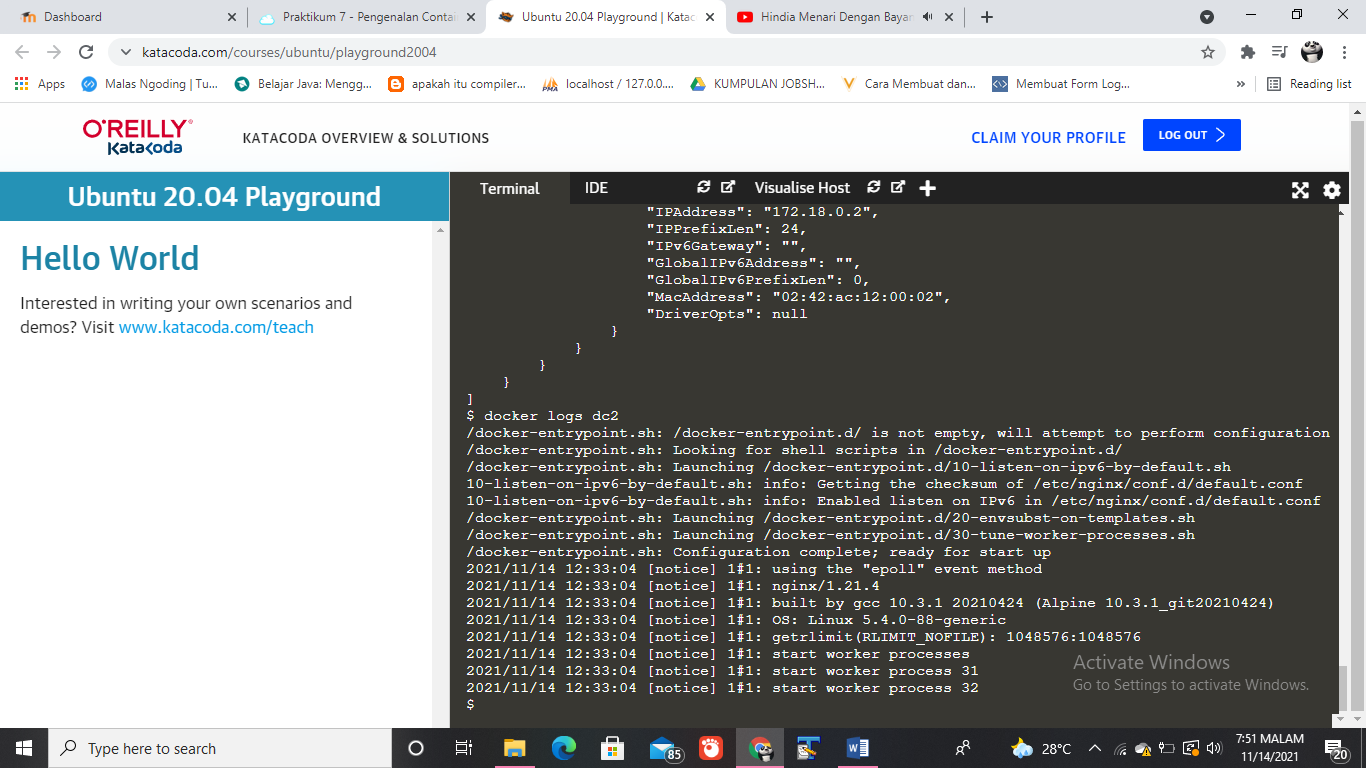
|  |
| --- |
| [  {  "Id": "dc2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524",  "Created": "2021-11-14T12:33:01.969839778Z",  "Path": "/docker-entrypoint.sh",  "Args": [  "nginx",  "-g",  "daemon off;"  ],  "State": {  "Status": "running",  "Running": true,  "Paused": false,  "Restarting": false,  "OOMKilled": false,  "Dead": false,  "Pid": 1483,  "ExitCode": 0,  "Error": "",  "StartedAt": "2021-11-14T12:33:04.339528931Z",  "FinishedAt": "0001-01-01T00:00:00Z"  },  "Image": "sha256:b46db85084b80a87b94cc930a74105b74763d0175e14f5913ea5b07c312870f8",  "ResolvConfPath": "/var/lib/docker/containers/dc2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524/resolv.conf",  "HostnamePath": "/var/lib/docker/containers/dc2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524/hostname",  "HostsPath": "/var/lib/docker/containers/dc2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524/hosts",  "LogPath": "/var/lib/docker/containers/dc2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524/dc2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524-json.log",  "Name": "/xenodochial\_kapitsa",  "RestartCount": 0,  "Driver": "overlay",  "Platform": "linux",  "MountLabel": "",  "ProcessLabel": "",  "AppArmorProfile": "docker-default",  "ExecIDs": null,  "HostConfig": {  "Binds": null,  "ContainerIDFile": "",  "LogConfig": {  "Type": "json-file",  "Config": {}  },  "NetworkMode": "default",  "PortBindings": {},  "RestartPolicy": {  "Name": "no",  "MaximumRetryCount": 0  },  "AutoRemove": false,  "VolumeDriver": "",  "VolumesFrom": null,  "CapAdd": null,  "CapDrop": null,  "CgroupnsMode": "host",  "Dns": [],  "DnsOptions": [],  "DnsSearch": [],  "ExtraHosts": null,  "GroupAdd": null,  "IpcMode": "private",  "Cgroup": "",  "Links": null,  "OomScoreAdj": 0,  "PidMode": "",  "Privileged": false,  "PublishAllPorts": false,  "ReadonlyRootfs": false,  "SecurityOpt": null,  "UTSMode": "",  "UsernsMode": "",  "ShmSize": 67108864,  "Runtime": "runc",  "ConsoleSize": [  0,  0  ],  "Isolation": "",  "CpuShares": 0,  "Memory": 0,  "NanoCpus": 0,  "CgroupParent": "",  "BlkioWeight": 0,  "BlkioWeightDevice": [],  "BlkioDeviceReadBps": null,  "BlkioDeviceWriteBps": null,  "BlkioDeviceReadIOps": null,  "BlkioDeviceWriteIOps": null,  "CpuPeriod": 0,  "CpuQuota": 0,  "CpuRealtimePeriod": 0,  "CpuRealtimeRuntime": 0,  "CpusetCpus": "",  "CpusetMems": "",  "Devices": [],  "DeviceCgroupRules": null,  "DeviceRequests": null,  "KernelMemory": 0,  "KernelMemoryTCP": 0,  "MemoryReservation": 0,  "MemorySwap": 0,  "MemorySwappiness": null,  "OomKillDisable": false,  "PidsLimit": null,  "Ulimits": null,  "CpuCount": 0,  "CpuPercent": 0,  "IOMaximumIOps": 0,  "IOMaximumBandwidth": 0,  "MaskedPaths": [  "/proc/asound",  "/proc/acpi",  "/proc/kcore",  "/proc/keys",  "/proc/latency\_stats",  "/proc/timer\_list",  "/proc/timer\_stats",  "/proc/sched\_debug",  "/proc/scsi",  "/sys/firmware"  ],  "ReadonlyPaths": [  "/proc/bus",  "/proc/fs",  "/proc/irq",  "/proc/sys",  "/proc/sysrq-trigger"  ]  },  "GraphDriver": {  "Data": {  "LowerDir": "/var/lib/docker/overlay/db3900e444ecc1acf68e4d376238d1d589deeb490b2015cb7518e0eb4668a5e4/root",  "MergedDir": "/var/lib/docker/overlay/196b7da3c870d20befe35ff775252ba6bb36290f308148c4d2ab4c9535bf1792/merged",  "UpperDir": "/var/lib/docker/overlay/196b7da3c870d20befe35ff775252ba6bb36290f308148c4d2ab4c9535bf1792/upper",  "WorkDir": "/var/lib/docker/overlay/196b7da3c870d20befe35ff775252ba6bb36290f308148c4d2ab4c9535bf1792/work"  },  "Name": "overlay"  },  "Mounts": [],  "Config": {  "Hostname": "dc2f1f945071",  "Domainname": "",  "User": "",  "AttachStdin": false,  "AttachStdout": false,  "AttachStderr": false,  "ExposedPorts": {  "80/tcp": {}  },  "Tty": false,  "OpenStdin": false,  "StdinOnce": false,  "Env": [  "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",  "NGINX\_VERSION=1.21.4",  "NJS\_VERSION=0.7.0",  "PKG\_RELEASE=1"  ],  "Cmd": [  "nginx",  "-g",  "daemon off;"  ],  "Image": "nginx:alpine",  "Volumes": null,  "WorkingDir": "",  "Entrypoint": [  "/docker-entrypoint.sh"  ],  "OnBuild": null,  "Labels": {  "maintainer": "NGINX Docker Maintainers <docker-maint@nginx.com>"  },  "StopSignal": "SIGQUIT"  },  "NetworkSettings": {  "Bridge": "",  "SandboxID": "d0d97315e62ebe4856192156fa4956f14c4e6e54c27e4a4a320677c93dca5114",  "HairpinMode": false,  "LinkLocalIPv6Address": "",  "LinkLocalIPv6PrefixLen": 0,  "Ports": {  "80/tcp": null  },  "SandboxKey": "/var/run/docker/netns/d0d97315e62e",  "SecondaryIPAddresses": null,  "SecondaryIPv6Addresses": null,  "EndpointID": "474a9a6b0c10293980cff08d9b16cb20127a93f63e66c91d8fced0b385ac2fa5",  "Gateway": "172.18.0.1",  "GlobalIPv6Address": "",  "GlobalIPv6PrefixLen": 0,  "IPAddress": "172.18.0.2",  "IPPrefixLen": 24,  "IPv6Gateway": "",  "MacAddress": "02:42:ac:12:00:02",  "Networks": {  "bridge": {  "IPAMConfig": null,  "Links": null,  "Aliases": null,  "NetworkID": "37f7e022d9808efe025920a58fe382dd4521fd06076c4755c1d7318e6640ba0a",  "EndpointID": "474a9a6b0c10293980cff08d9b16cb20127a93f63e66c91d8fced0b385ac2fa5",  "Gateway": "172.18.0.1",  "IPAddress": "172.18.0.2",  "IPPrefixLen": 24,  "IPv6Gateway": "",  "GlobalIPv6Address": "",  "GlobalIPv6PrefixLen": 0,  "MacAddress": "02:42:ac:12:00:02",  "DriverOpts": null  }  }  }  }  ] |

Sementara untuk melihat catatan sistem log untuk container anda dapat menggunakan perintah :

docker logs <container id / friendly name>

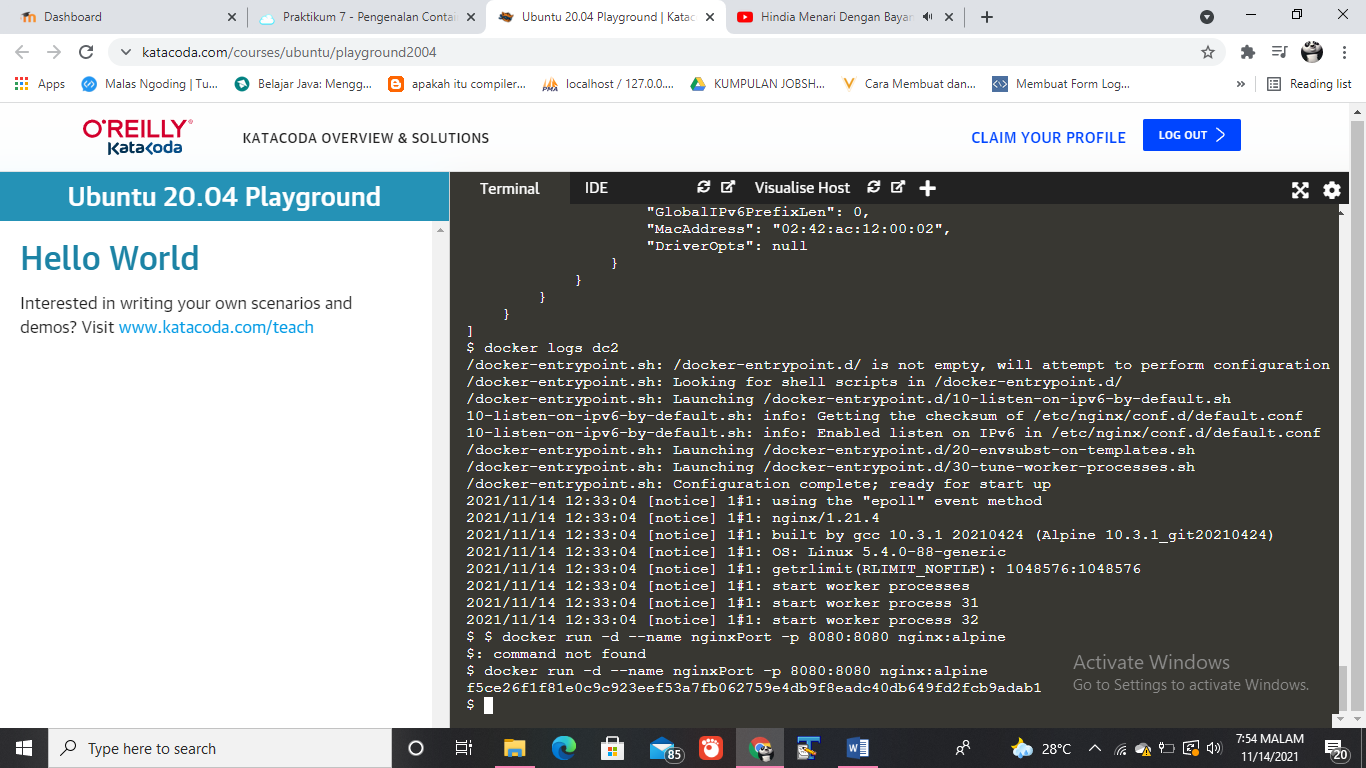
Perintah ini digunakan untuk proses troubleshooting ketika layanan didalam container tidak berjalan atau ada problem pada container. Perintah yang digunakan secara lengkap adalah sebagai berikut (asumsi masih menggunakan container ID yang sama dg perintah sebelumnya):

docker logs dc2

Hasil output :

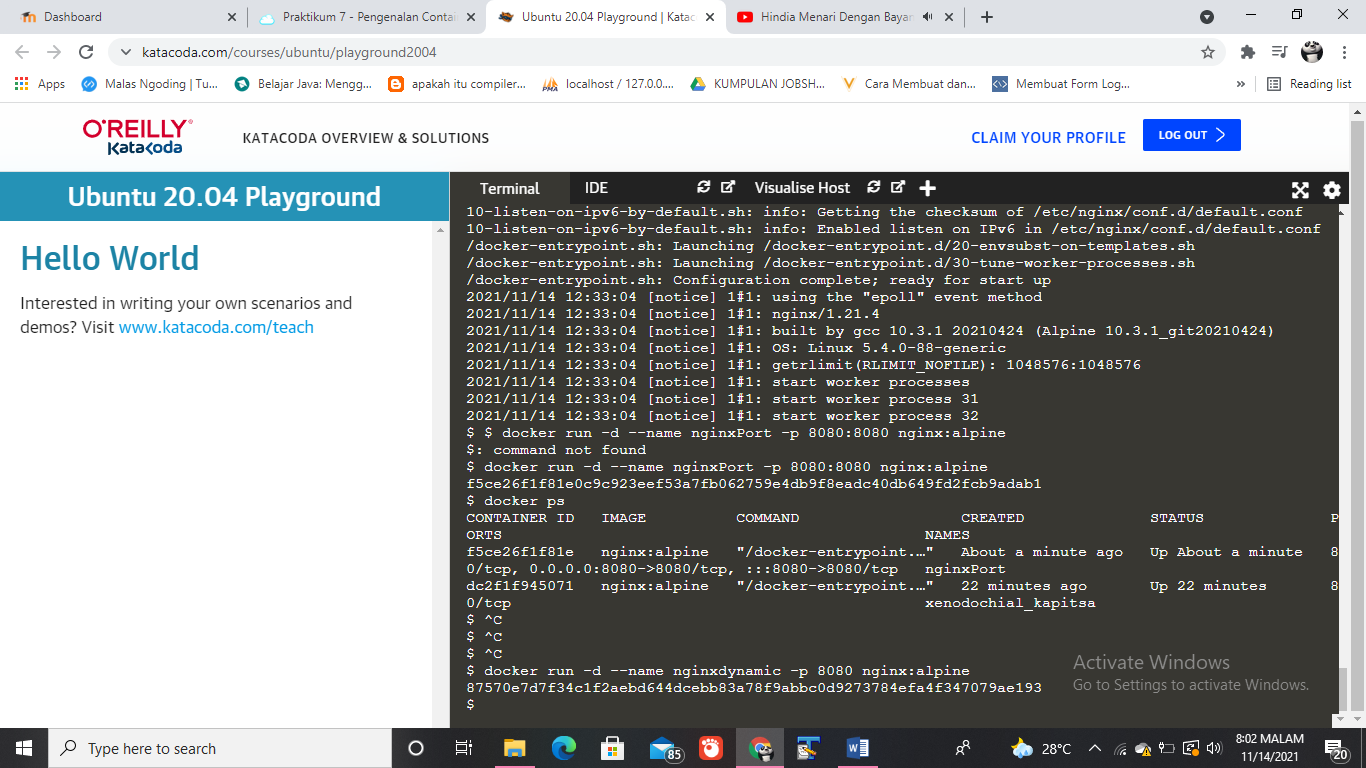
Jika ingin menggunakan perintah inspect atau logs pada docker, anda perlu tahu id container atau nama dari container. Sebelum menjalankan perintah logs atau inspect, jalankan dulu perintah **docker ps.**

1. Binding Port

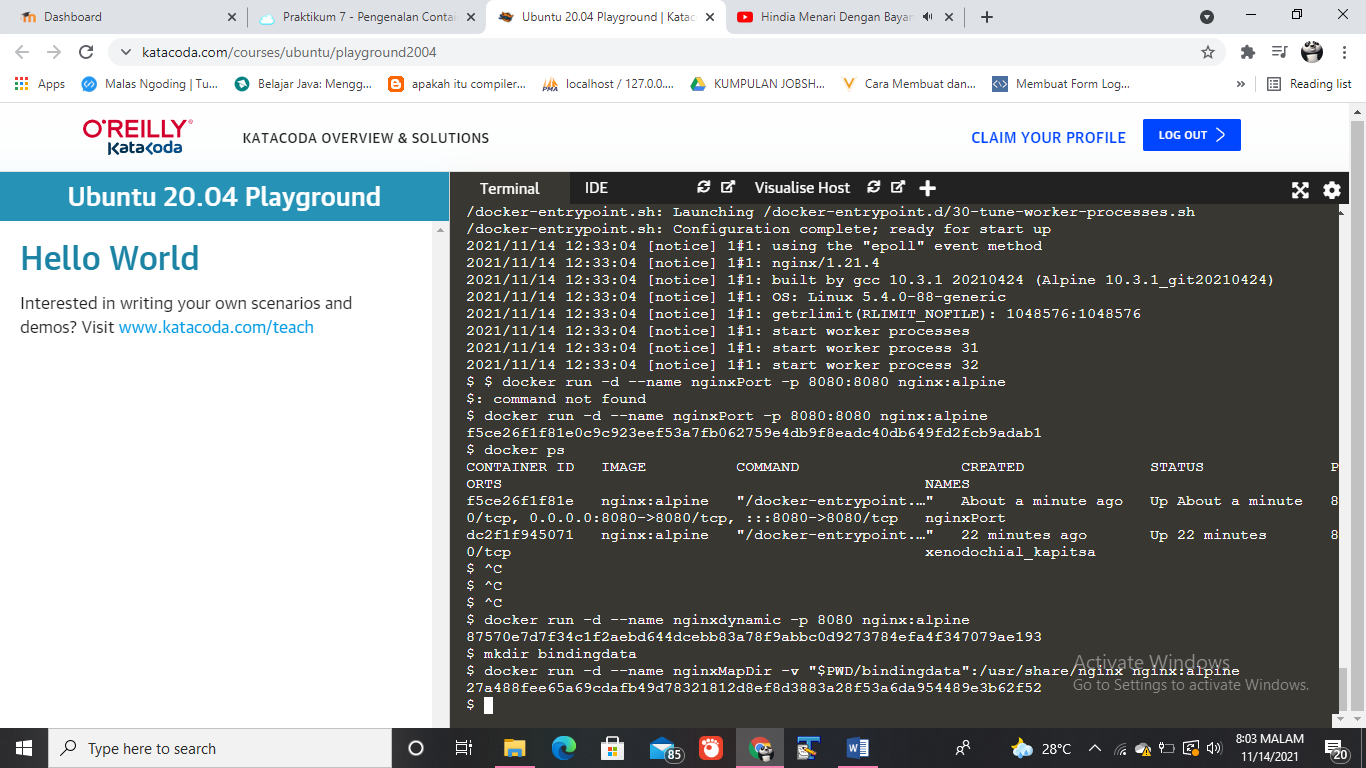
Container yang berjalan terkadang perlu untuk dapat di akses secara eksternal, untuk memudahkan Docker dapat melakukan binding port untuk host dan container. Dalam contoh dibawah ini, melakukan binding pada port <host port>:<container port> dengan menggunakan opsi -p. --name adalah opsi untuk memberikan friendly name (tag) pada container, sementara itu nginx:alpine adalah image yang digunakan untuk membuat container. Dibawah ini adalah contoh implementasi perintahnya

Setelah berjalan, anda dapat melakukan pengecekan dengan melihat proses dari Docker. Pada hasil dibawah, terlihat ada 2 proses container yang sedang berjalan. Jika menggunakan perintah docker ps, akan menampilkan seperti berikut:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAME |
| f5ce26f1f81e | nginx:alpine | "/docker-entrypoint.…" | About a minute ago | Up About a minute | 80/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp | Nginx  Port |
| dc2f1f945071 | nginx:alpine | "/docker-entrypoint.…" | 22 minutes ago | Up 22 minutes | 80/tcp | xenodochial\_kapitsa |

Docker juga memungkinkan container dapat berjalan sekaligus pada 1 lingkungan host, sehingga container port akan di konfigurasi secara default. Berikut contoh implementasinya:

1. Binding Data

Containers secara desain bersifat stateless. Seluruh data yang kita inginkan untuk tetap ada setelah container berhenti dijalankan harus tersimpan pada mesin host. Untuk mengatasi persoalan ini, dapat dilakukan dengan cara binding host direktori ke dalam container. Dibawah ini adalah contoh implementasi:

1. Membuat Docker Images

Docker memungkinkan untuk membuat image sendiri dengan base Docker image yang tersedia. Untuk membuat diperlukan file Dockerfile. Berikut ini langkah untuk membuat image tersebut, dalam contoh ini, seluruh file kebutuhan diletakkan pada direktori imageku. Buat direktori imageku dengan perintah:

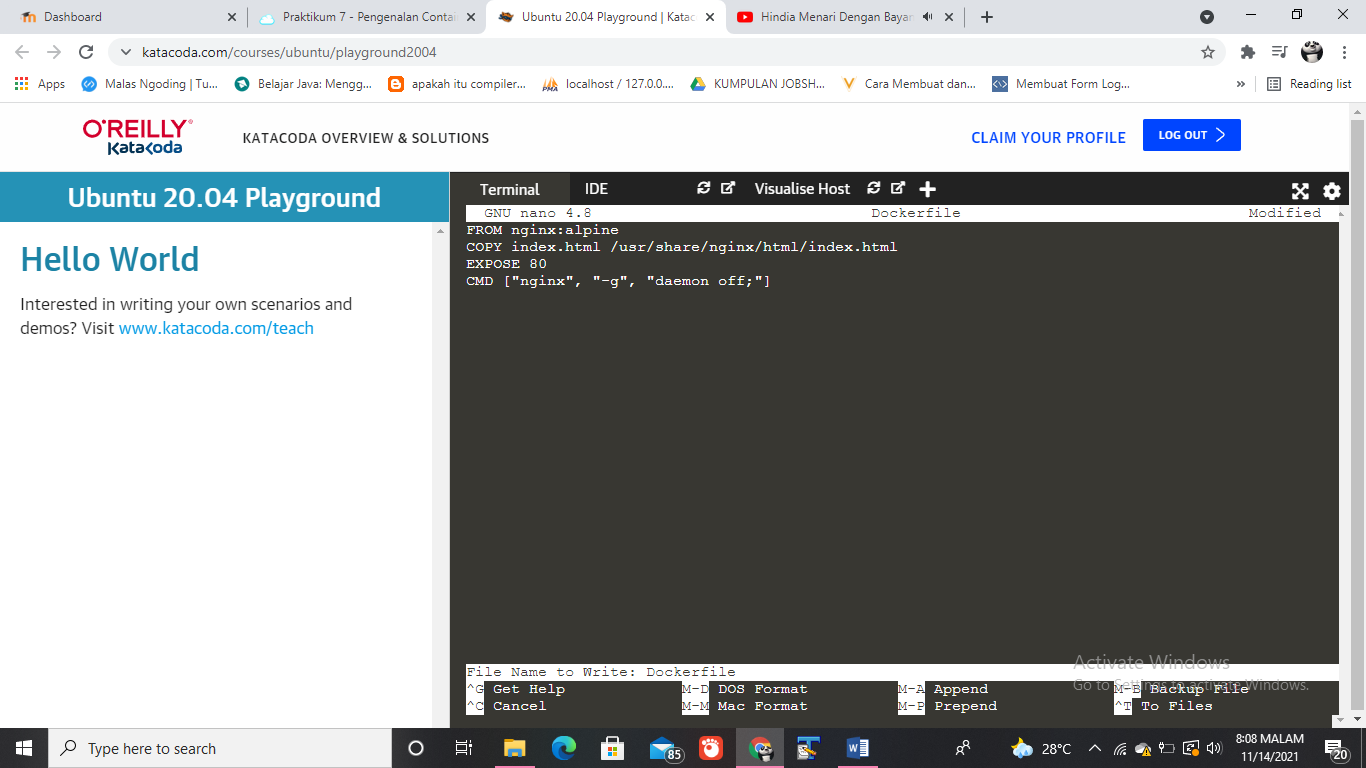
mkdir imageku

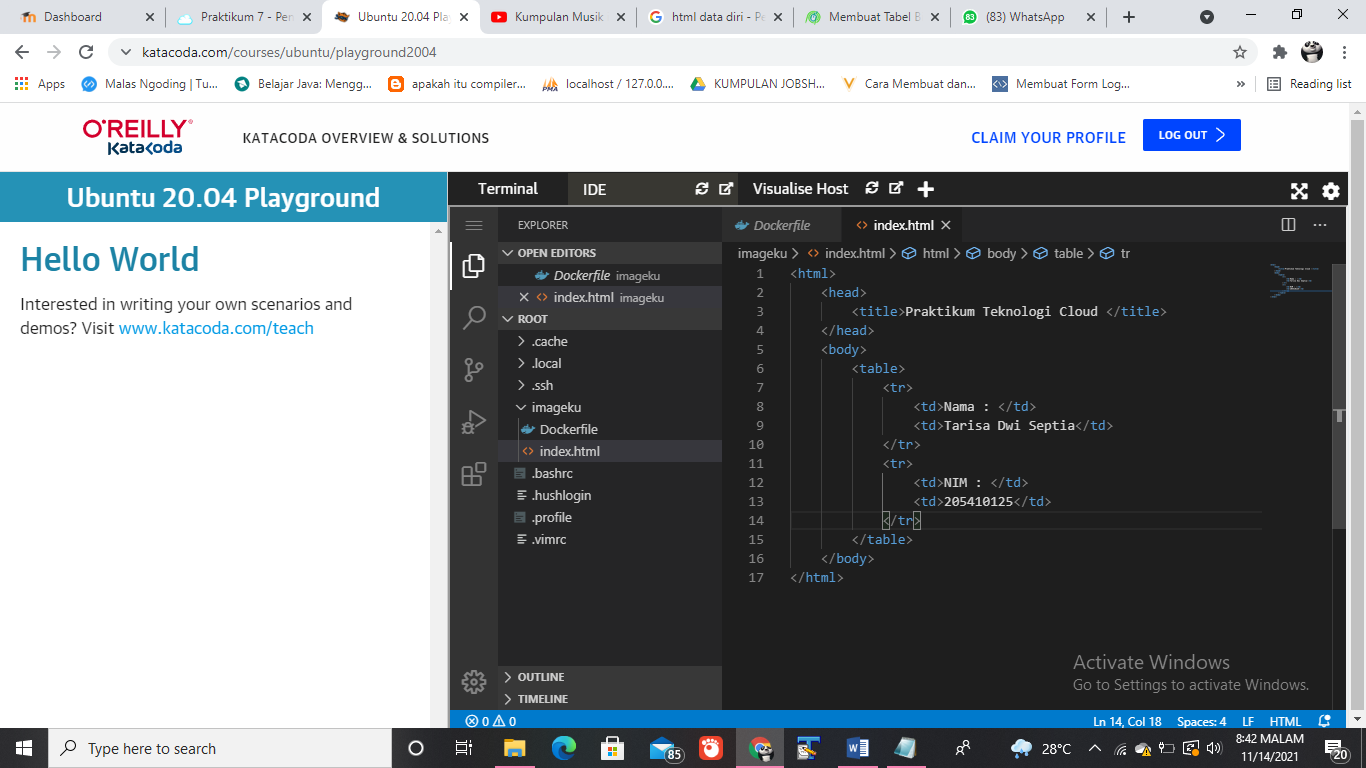
Masuk ke dalam direktori tersebut, dengan perintah:

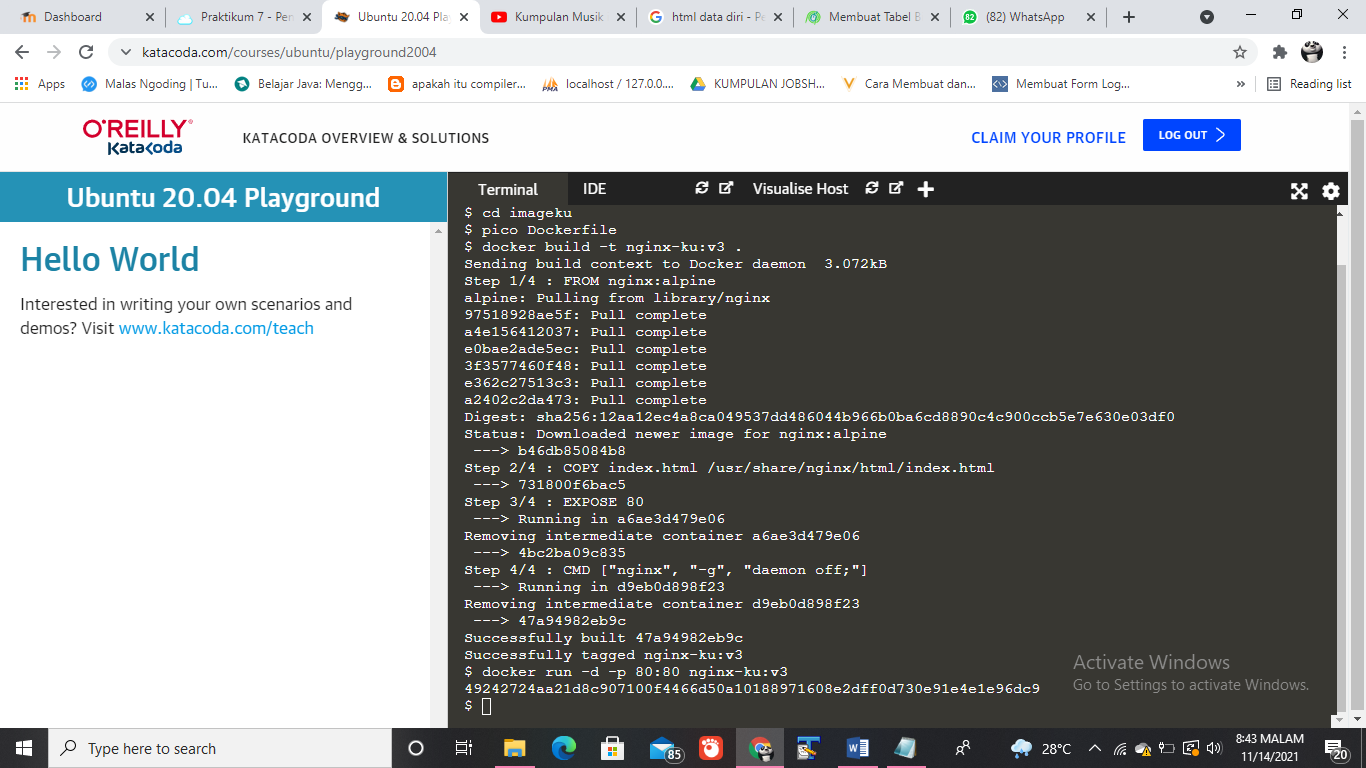
cd imageku

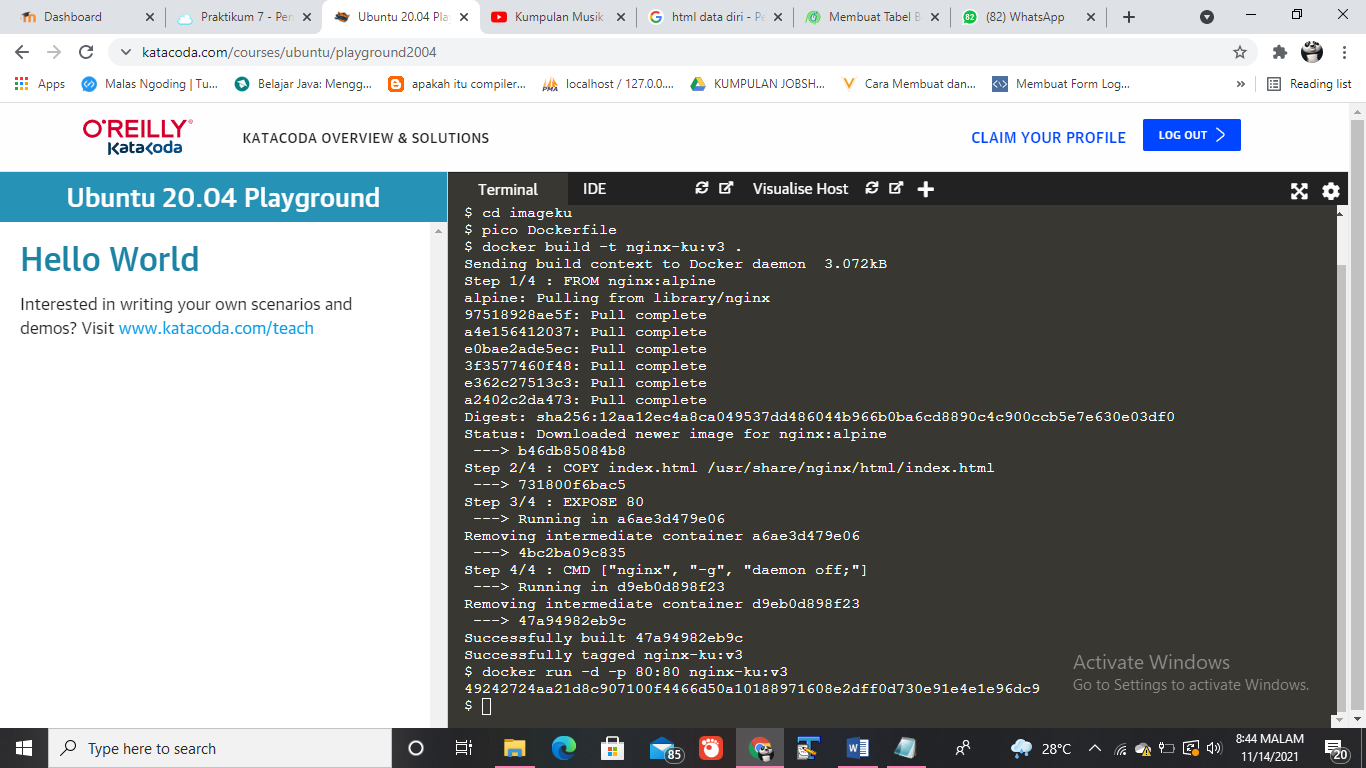
Lalu gunakan text editor pico untuk membuat file bernama Dockerfile :

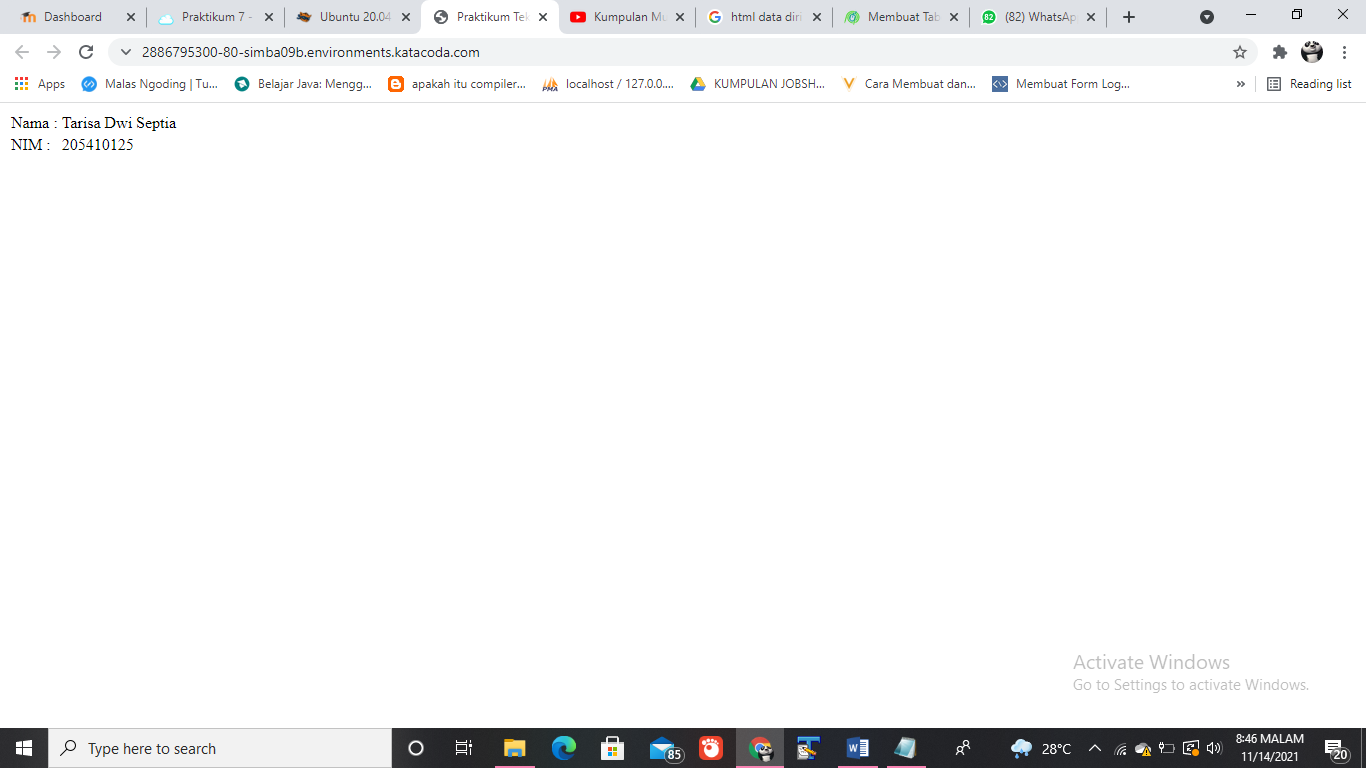
pico Dockerfile

Kemudian copy paste baris code berikut

Selanjutnya, buatlah file html sederhana bernama index.html untuk menampilkan NAMA, NIM, dan DESKRIPSI DIRI.

Setelah seluruh file tersebut tersedia, anda dapat membuat image dengan format penamaan nama-aplikasi:versi, contoh nginx-ku:v3 :

Langkah terakhir, image tersebut dapat anda jalankan sebagai container.

Hasil :

1. **Kesimpulan**

Setelah melakukan praktik diatas sesuai dengan tujuan pada modul 7 ini, dapat disimpulkan bahwa mahasiswa dapat mempelajari dan mengimplementasikan container docker