TASK 1

1. Understand Docker Networking Basics

Research the default Docker network types: bridge, host, none, and overlay.

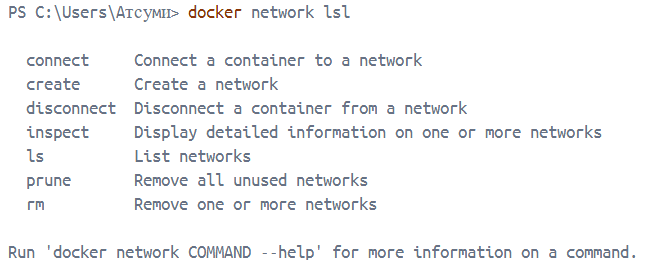
Learn about container-to-container communication and how Docker uses IP addresses and network namespaces.

У Docker є кілька основних типів мереж:

* bridge – створюється за замовчуванням і дозволяє контейнерам спілкуватися один з одним.
* host – контейнер використовує мережу хоста без ізоляції.
* none – контейнер без мережі.
* overlay – використовується для з'єднання контейнерів між різними хостами (у Swarm-режимі).

2. List Available Networks

Run the following command to list existing Docker networks: docker network ls



3. Inspect a Network

Choose one network from the list and inspect its configuration: docker network inspect

<network\_name>

Обираємо мережу зі списку, до прикладу, bridge, та пишемо наступну команду:

docker network inspect bridge

Це покаже її детальну конфігурацію.

PS C:\Users\Атсуми> docker network inspect bridge

[

{

"Name": "bridge",

"Id": "3acbff6fd461e57058c7fb9ead053da53c0eb11ab044974dcb315bed8126dc94",

"Created": "2025-03-19T15:56:24.246763132Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "172.17.0.0/16",

"Gateway": "172.17.0.1"

}

]

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {},

"Options": {

"com.docker.network.bridge.default\_bridge": "true",

"com.docker.network.bridge.enable\_icc": "true",

"com.docker.network.bridge.enable\_ip\_masquerade": "true",

"com.docker.network.bridge.host\_binding\_ipv4": "0.0.0.0",

"com.docker.network.bridge.name": "docker0",

"com.docker.network.driver.mtu": "1500"

},

"Labels": {}

}

]

4. Create a Custom Network

Create a bridge network named my\_custom\_network: docker network create

my\_custom\_network

Створимо нову мережу my\_custom\_network та перевіримо чи з'явилася вона у списку.

PS C:\Users\Атсуми> docker network create my\_custom\_network

8029fb1dc077130d6716aa183df0b9a9d326b0089fe1b8a77c9f2bc691efcb9f

PS C:\Users\Атсуми> docker network ls

NETWORK ID NAME DRIVER SCOPE

3acbff6fd461 bridge bridge local

9e205c5266f6 host host local

8029fb1dc077 my\_custom\_network bridge local

106417cd3727 none null local

5. Run Containers in the Custom Network

Start two containers (nginx and alpine) and connect them to my\_custom\_network:

docker run -dit --name nginx --network my\_custom\_network nginx

docker run -dit --name alpine --network my\_custom\_network alpine sh

Запустимо два контейнери – nginx (веб-сервер) і alpine (ОС) та перевіримо, що вони працюють: docker ps

PS C:\Users\Атсуми> docker run -dit --name nginx --network my\_custom\_network nginx

Unable to find image 'nginx:latest' locally

latest: Pulling from library/nginx

c22eb46e871a: Download complete

417c4bccf534: Download complete

373fe654e984: Download complete

97f5c0f51d43: Download complete

e7e0ca015e55: Download complete

5eaa34f5b9c2: Download complete

6e909acdb790: Download complete

Digest: sha256:124b44bfc9ccd1f3cedf4b592d4d1e8bddb78b51ec2ed5056c52d3692baebc19

Status: Downloaded newer image for nginx:latest

e6a03443fdb12a8478bd853258f9909545ad28b1c9d092bec0b86ed2b8f33d64

PS C:\Users\Атсуми> docker run -dit --name alpine --network my\_custom\_network alpine sh

Unable to find image 'alpine:latest' locally

latest: Pulling from library/alpine

f18232174bc9: Download complete

Digest: sha256:a8560b36e8b8210634f77d9f7f9efd7ffa463e380b75e2e74aff4511df3ef88c

Status: Downloaded newer image for alpine:latest

9f95bb6c5454dfeeab6f35740b8d0c38e4ea41703bba49d07ed845b19bf5b849

PS C:\Users\Атсуми> docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

9f95bb6c5454 alpine "sh" 16 seconds ago Up 15 seconds alpine

e6a03443fdb1 nginx "/docker-entrypoint.…" 34 seconds ago Up 33 seconds 80/tcp nginx

6. Test Connectivity

Access the alpine container and test connectivity with the nginx container:

docker exec -it alpine sh

ping nginx

apk add curl # Install curl command

curl <http://nginx>

PS C:\Users\Атсуми> docker exec -it alpine sh

/ # apk add curl

fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/main/x86\_64/APKINDEX.tar.gz

fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/community/x86\_64/APKINDEX.tar.gz

(1/9) Installing brotli-libs (1.1.0-r2)

(2/9) Installing c-ares (1.34.3-r0)

(3/9) Installing libunistring (1.2-r0)

(4/9) Installing libidn2 (2.3.7-r0)

(5/9) Installing nghttp2-libs (1.64.0-r0)

(6/9) Installing libpsl (0.21.5-r3)

(7/9) Installing zstd-libs (1.5.6-r2)

(8/9) Installing libcurl (8.12.1-r1)

(9/9) Installing curl (8.12.1-r1)

Executing busybox-1.37.0-r12.trigger

OK: 12 MiB in 24 packages

/ # curl https://nginx

curl: (7) Failed to connect to nginx port 443 after 0 ms: Could not connect to server

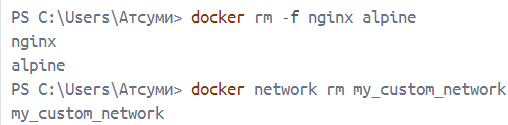
7. Clean Up

Stop and remove the containers:

docker rm -f nginx alpine

Remove the custom network:

docker network rm my\_custom\_network



TASK 2

1. \*Run at least two containers (backend and database) using dockercompose. Backend should connect to database.

PS C:\Users\Атсуми> docker --version

Docker version 27.5.1, build 9f9e405

PS C:\Users\Атсуми> docker-compose --version

Docker Compose version v2.32.4-desktop.1

Створюємо файл docker-compose.yml в C:\Users\Ім'я\my\_project\docker-compose.yml

Додаємо наступний код у файл:

version: '3.8'

services:

database:

image: postgres:15

container\_name: postgres\_db

restart: always

environment:

POSTGRES\_USER: user

POSTGRES\_PASSWORD: password

POSTGRES\_DB: mydb

ports:

- "5432:5432"

networks:

- my\_network

backend:

image: python:3.9

container\_name: backend\_app

restart: always

depends\_on:

- database

networks:

- my\_network

command: >

sh -c "pip install psycopg2 && python -c 'import psycopg2; conn = psycopg2.connect(\"dbname=mydb user=user password=password host=database\"); print(\"Connected to DB!\")'"

networks:

my\_network:

driver: bridge

Переходимо у папку з файлом docker-compose.yml у терміналі Docker Desktop

cd /Users/Атсуми/my\_project

Запустимо docker-compose:

docker compose up -d

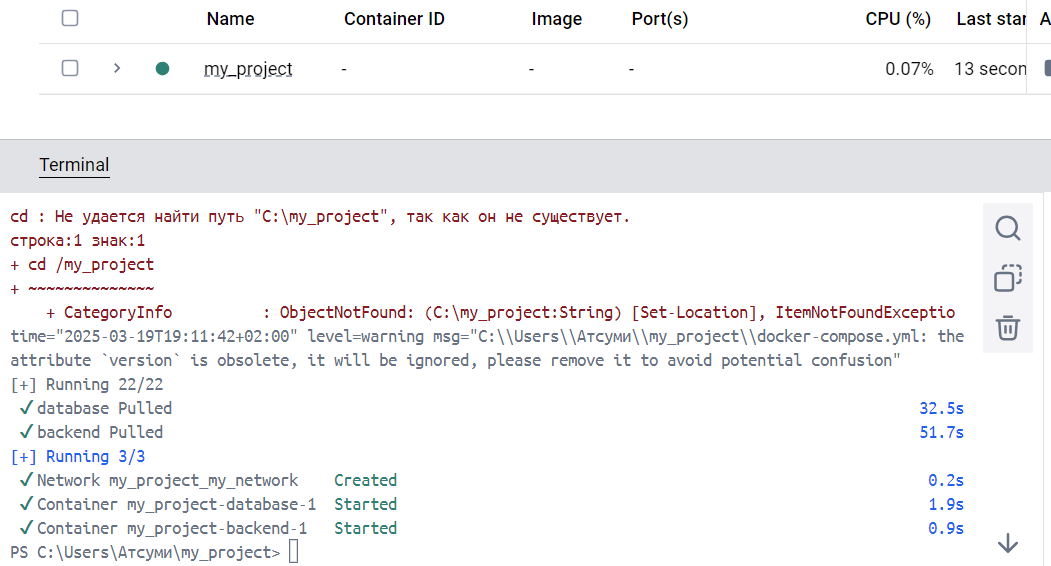
Перевіряємо роботу контейнерів

docker ps

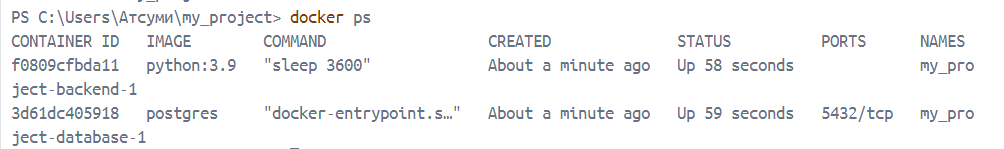
Щоб побачити повідомлення "Connected to DB!":

docker logs backend\_app





Перевірка роботи контейнерів:



Зупинка і видалення контейнерів

docker compose down