DISTRIBUTED AND CLOUD COMPUTING

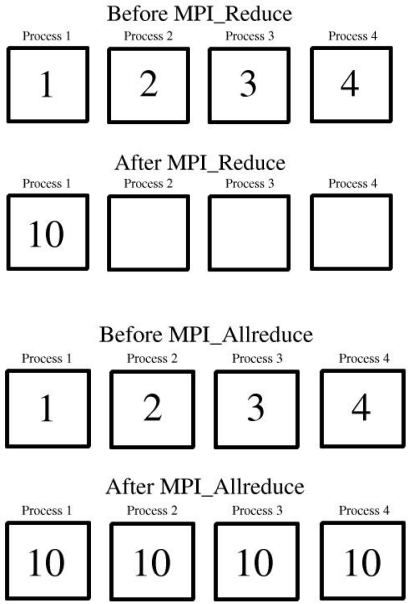
LAB 3: MPI REDUCE AND DOCKER

COLLECTIVE COMMUNICATION (N-N)

Reduce methods implement a distributed computation using data distributed over all processes

MPI_Reduce: result of computation is gathered by the ROOT

MPI_Allreduce: result of computation is broadcasted to all processes



BUILT-IN OPERATIONS IN MPI

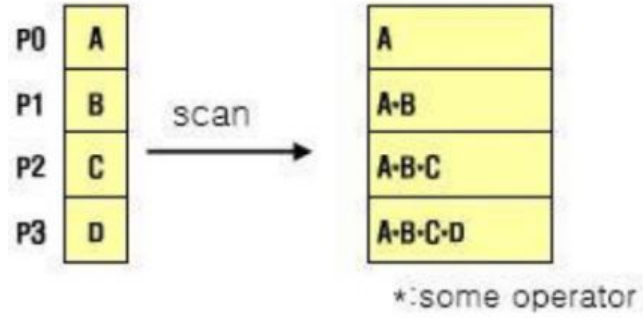
[MPI_MAX] maximum
[MPI_MIN] minimum
[MPI_SUM] sum
[MPI_PROD] product
[MPI_LAND] logical and
[MPI_BAND] bit-wise and
[MPI_LOR] logical or
[MPI_BOR] bit-wise or
[MPI_LXOR] logical xor
[MPI_BXOR] bit-wise xor
[MPI_MAXLOC] max value and location
[MPI_MINLOC] min value and location

https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/

COLLECTIVE COMMUNICATION (N-N)

Scan: Each process get the first "rank" items

- process 1 gets the first item
- process 2 gets the first two items

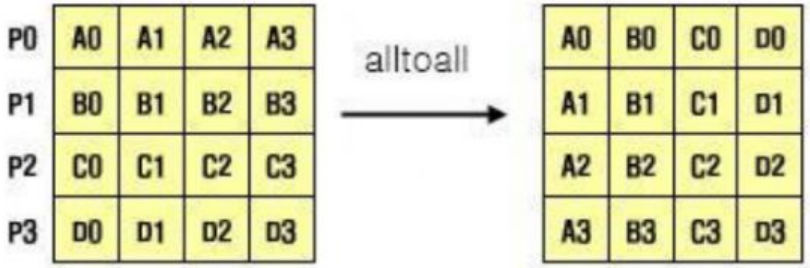


AlltoAll: Each process gets **all** "rank" items

- process 1 gets all the items with index 1
- process 2 gets all items with index 2

Alternative explanation:

It's a transpose operation!

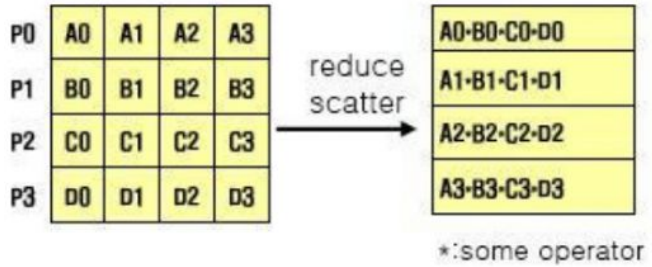


COLLECTIVE COMMUNICATION (N-N)

MPI_Reduce_scatter: scatters data and perform reduction

The 'scatter' operation is an **AllToAll operation!**

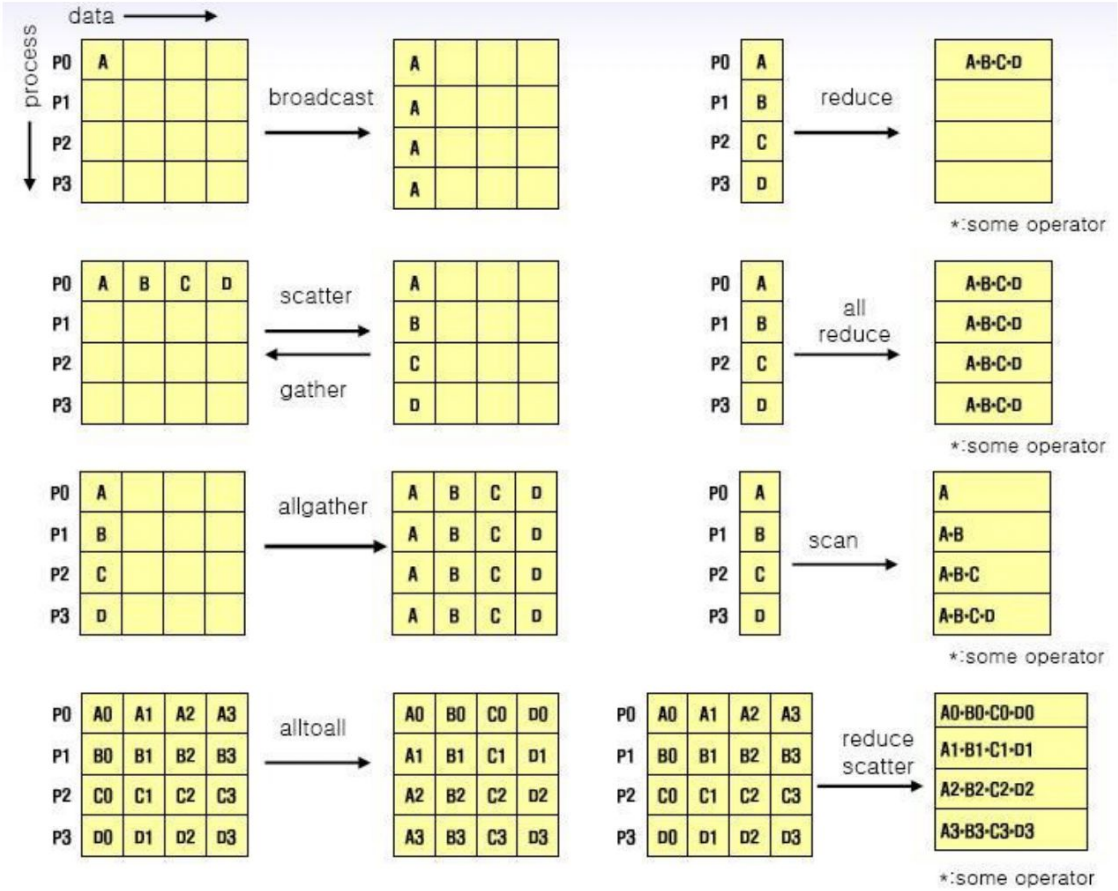
If it helps: you can remember this as 'Transpose Reduce'



BUILT-IN OPERATIONS IN MPI

[MPI_MAX] maximum
[MPI_MIN] minimum
[MPI_SUM] sum
[MPI_PROD] product
[MPI_LAND] logical and
[MPI_BAND] bit-wise and
[MPI_LOR] logical or
[MPI_BOR] bit-wise or
[MPI_LXOR] logical xor
[MPI_BXOR] bit-wise xor
[MPI_MAXLOC] max value and location
[MPI_MINLOC] min value and location

Collective communication in one figure



TASK: DISTRIBUTED DOT PRODUCT

PART A: Study the provided MPI examples for N-to-N collective communication

Source files: mpi_NtoN_example.c | mpi_reduce_scatter_example.c | mpi_scan_example.c

PART B: Using 3 workers perform a dot product operation on 2 vectors of length 9

- **Step 1:** Root (0) scatters the vectors over the workers (including self)
- **Step 2:** Workers perform dot product on sub-vectors
- **Step 3:** Perform a reduction on dot products (MPI_SUM) and print result on root (0)

```
mpicc source.c -o executable_name
mpirun –np <num_processes>  ./executable_name
```

DISTRIBUTED DOT PRODUCT (observations)

VECTOR SIZE: 800

```
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 1 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000013 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 1 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000007 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 1 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000007 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 4 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000246 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 4 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000130 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 4 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000110 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 8 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000134 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 8 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000151 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 8 vect --size 800
ROOT received the reduced dot product: 170346800
Execution time: 0.000187 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
```

VECTOR SIZE: 500,000,000

```
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 1 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 5.305913 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 1 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 3.753847 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 1 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 3.670115 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 4 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 3.885150 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 4 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 3.023296 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 4 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 3.082895 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 8 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 3.882954 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 8 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 3.509453 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$ mpirun -n 8 vect --size 500000000
ROOT received the reduced dot product: 9578198922199153536
Execution time: 3.496128 seconds
george@DESKTOP-E24BUDU:~/DistSys24/MPI/w3$
```

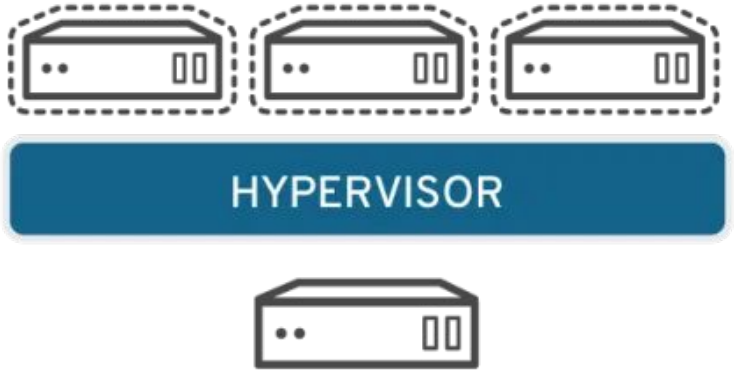
DISTRIBUTED DOT PRODUCT (observations)

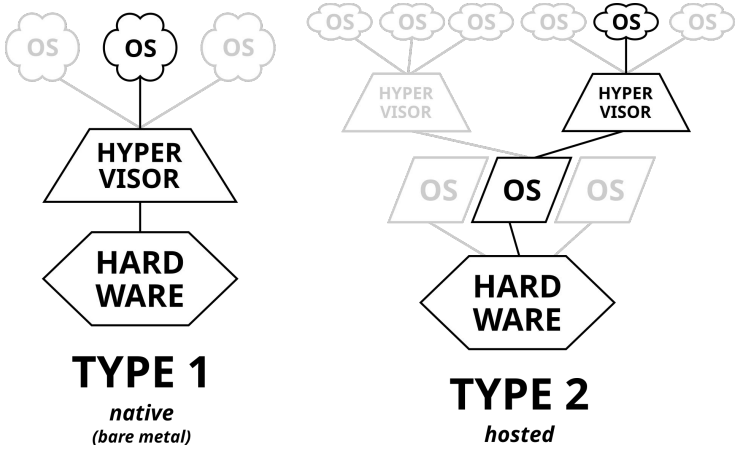
- **There is a TRADE-OFF!**
- The more processes we have the more communication is required
- If COMPUTATION GAIN **<** COMMUNICATION COST
  - The program will run **SLOWER** with more processes!

https://aws.amazon.com/what-is/virtualization/
https://www.redhat.com/en/topics/virtualization/what-is-virtualization
https://en.wikipedia.org/wiki/Hypervisor

Virtualization: Enabling Cloud Computing

- **Virtualization**:
  a. creates virtual representations of hardware
  b. powers cloud computing services by helping organizations manage infrastructure
- **Why?**
  a. More flexible & secure resource utilisation. Isolation
  b. Painless environment setup & recovery effort (keep everything tidy)
- **Virtual Machine Monitor (VMM, or Hypervisor)**: a type of computer software, wor hardware that creates and runs **virtual machines (VMs)**.

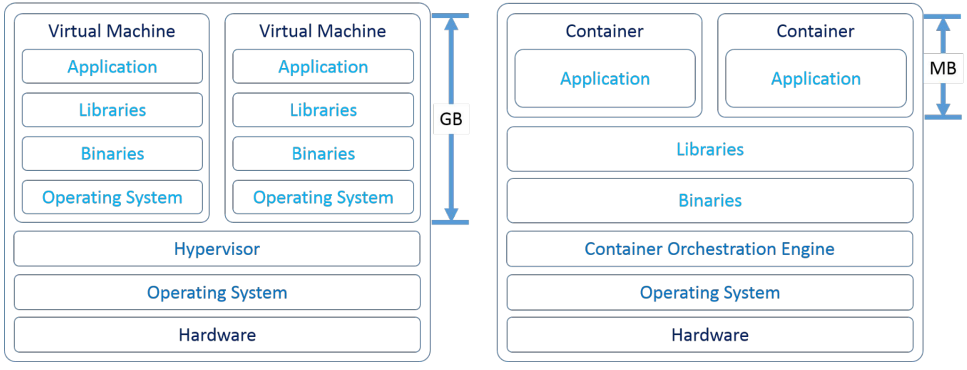


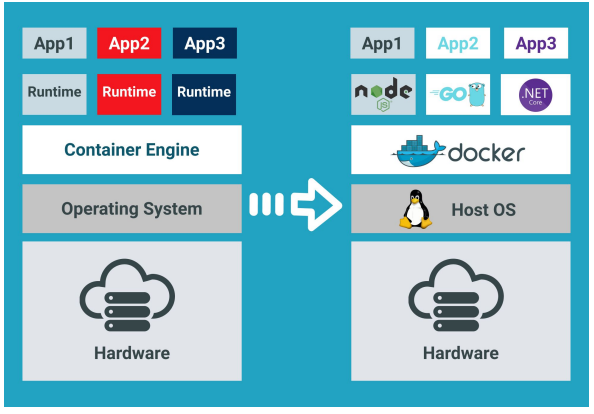


https://cloud.google.com/learn/what-are-containers
https://www.alibabacloud.com/en/knowledge/difference-between-container-and-virtual-machine
https://dbconvert.com/blog/content/images/size/w1600/2021/02/docker-apps.png

Containerization

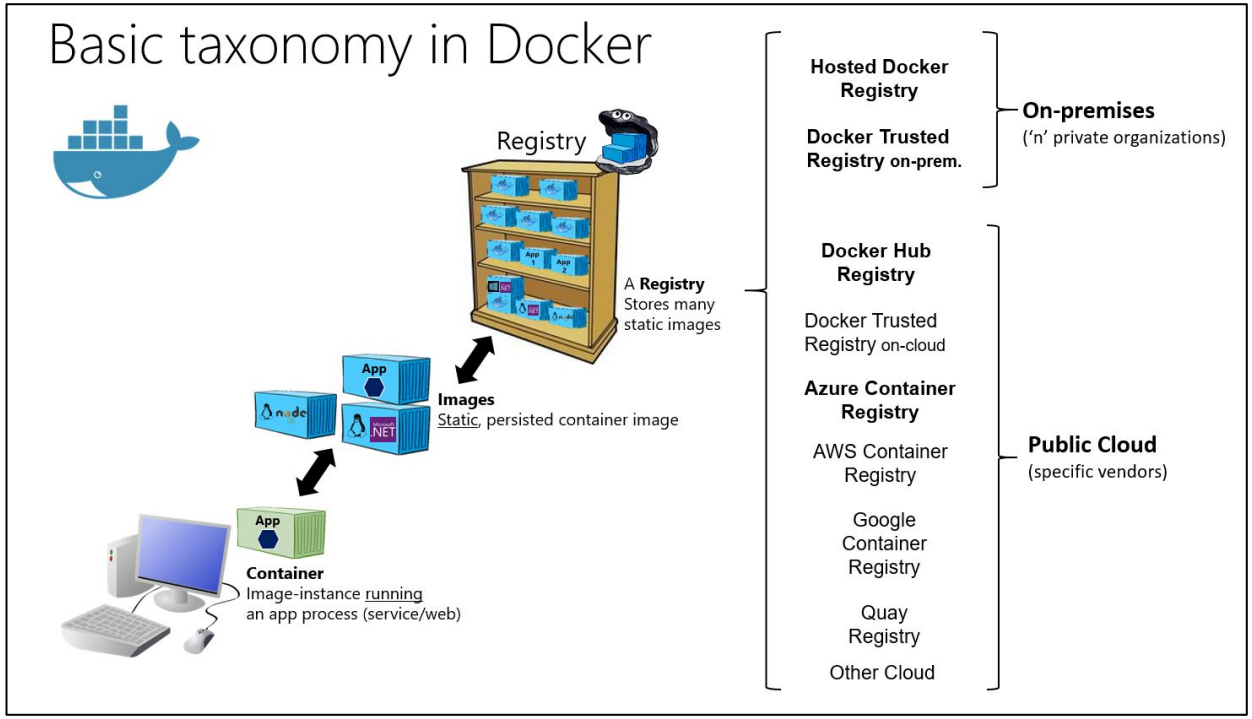
- **Containers**: lightweight packages of application code together with dependencies required to run your software services.
- **Containers - compared to VMs**:
  a. **Efficient**:
     - containerized apps share the hosting OS kernel, thus using fewer resources.
     - even easier environment setup.
  b. **Lightweight**: do not carry OS; only contain necessary libraries & tools.
  c. **Portability**: OS-independent, thus movable to PCs/VMs/Cloud.





Docker: Basic Concepts

- **Registry**
  - Docker Hub
- **Image**
  - OS like ubuntu
  - apps like postgres
- **Container**
  - image instance

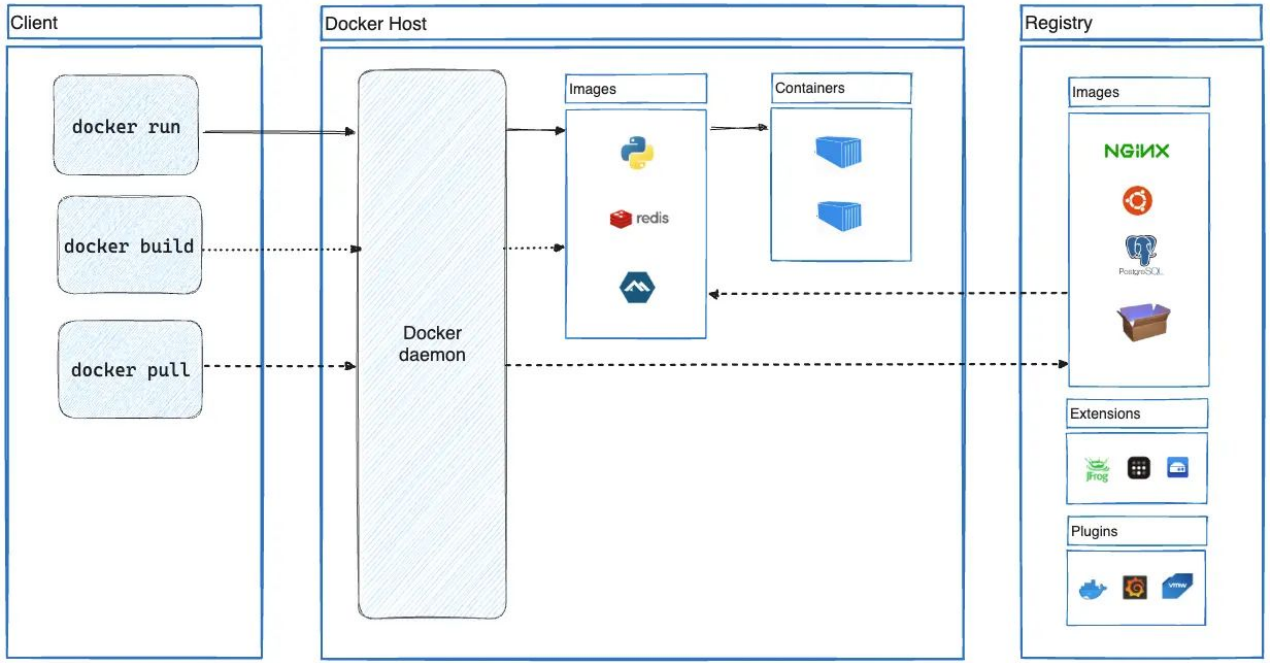


https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-containers-images-registries
ubuntu image: https://hub.docker.com/_/ubuntu
postgres image: https://hub.docker.com/_/postgres

Docker: Architecture

- **Registry**
  - pull images
  - `docker pull`
- **Image**
  - build images
  - requires `Dockerfile`
  - `docker build`
- **Container**
  - run images
  - `docker run`

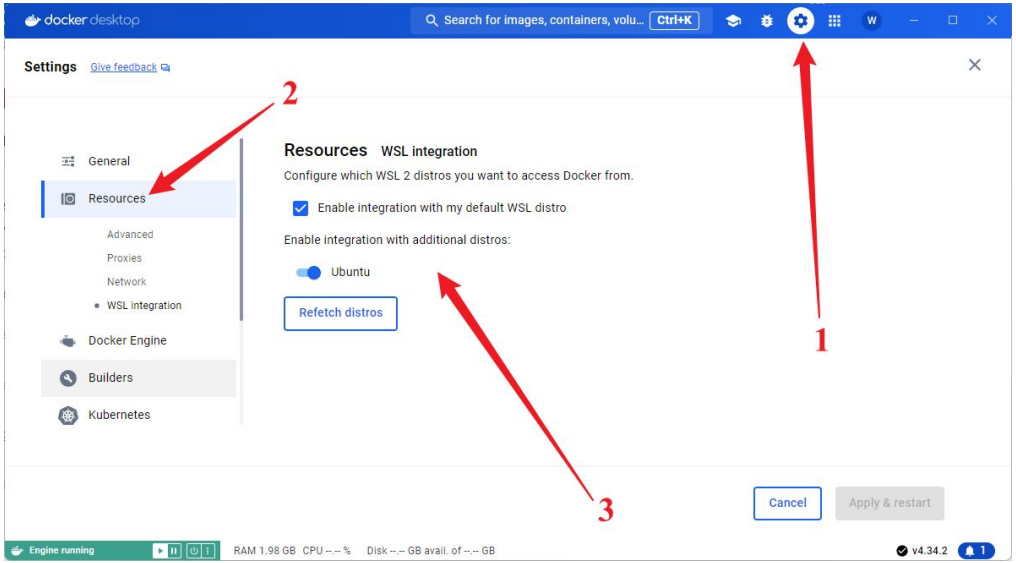
- **Docker CLI/Client**
  - `docker` commands
- **Docker daemon**
  - middleman between Docker Client & docker objects
- **Docker Desktop**
  - includes the above

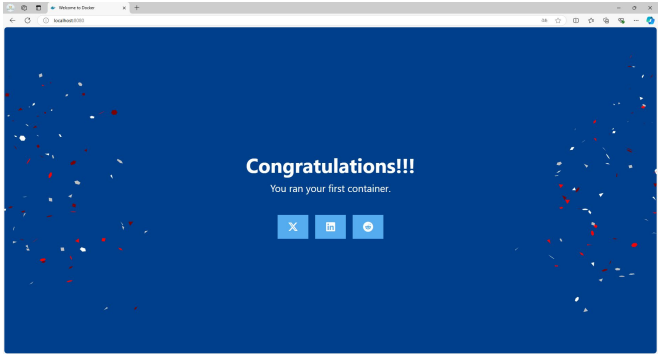


https://docs.docker.com/get-started/docker-overview/#docker-architecture

Docker: Setup

1. Install Docker Desktop: https://docs.docker.com/get-started/introduction/get-docker-desktop/.
2. For WSL2 users: enable WSL integration in Docker Desktop settings.
3. Verify installation by: `docker run -d -p 8080:80 docker/welcome-to-docker`
4. Verify installation by accessing: http://localhost:8080/.

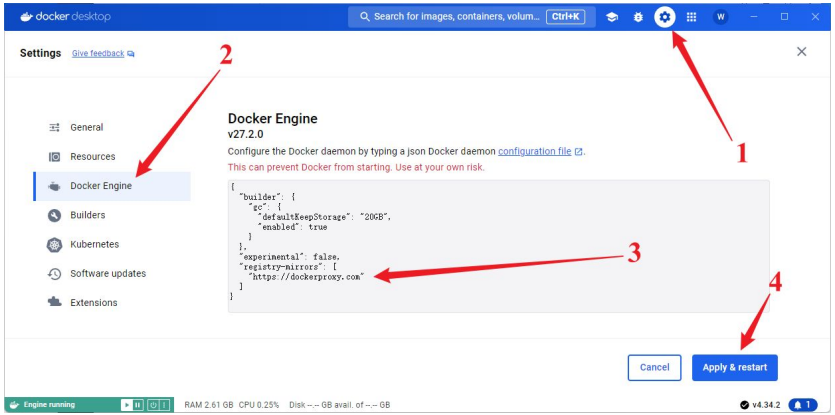




Docker: Setup

(optional) If image pulling encounters timeouts, try to set registry mirrors in the settings: add https://dockerproxy.com as a registry mirror:

```
"registry-mirrors": [
  "https://dockerproxy.com"
]
```



After verification, stop the container and clean the resources.

1. Find the corresponding container ID/name: `docker ps`
2. Stop the container: `docker container stop [ID/NAME]`
3. Remove the container: `docker container rm [ID/NAME]`

```
(base) root@RAINBOW: # docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED          STATUS          PORTS                    NAMES
84566a2f3e78   docker/welcome-to-docker "/docker-entrypoint.…"   9 minutes ago    Up 9 minutes    0.0.0.0:8080->80/tcp     sharp_gould
```

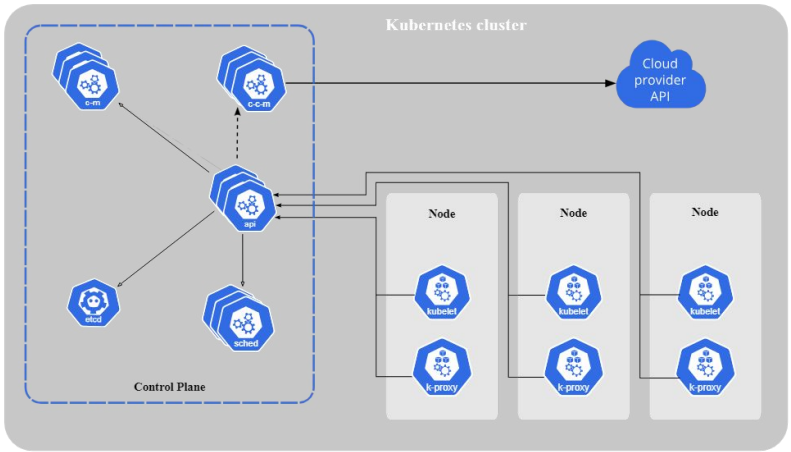
copy this ID

or copy this name

```
(base) root@RAINBOW: # docker container stop 84566a2f3e78
84566a2f3e78
(base) root@RAINBOW: # docker container rm 84566a2f3e78
84566a2f3e78
```

Docker Compose, Swarm & Kubernetes

- **Docker Compose**: a tool for defining and running multi-container applications on a single host.
  - Requires `compose.yaml`.
- **Docker - Swarm mode**: an advanced feature for managing a cluster of Docker daemons - running multi-container applications on multiple hosts.
- **Kubernetes (K8s)**: a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation (more details in the future).



https://docs.docker.com/compose/
https://docs.docker.com/engine/swarm/
https://kubernetes.io/docs/concepts/

TASK: Test MPI with Docker Compose

Download the sample code zip file from Blackboard and try to run yourself. This sample executes the MPI Hello World program we wrote previously on 2 containers via Docker Compose. It contains:

- a built executable MPI program: `intro`
- `Dockerfile` for `docker build`
- `compose.yaml` for `docker compose`

```
(base) root@RAINBOW:/home/rainbow/asialab/docker_dist_mpi# ls -lhR
.:
total 12K
-rw-rw-rw- 1 root root 1.2K Sep 24 14:14 Dockerfile
-rw-rw-rw- 1 root root  977 Sep 24 14:10 compose.yaml
drwxrwxrwx 2 root root 4.0K Sep 24 14:10 mpi_files

./mpi_files:
total 16K
-rwxrwxrwx 1 root root 16K Sep 11 10:53 intro
```

Follow the following steps:

1. Build an image named `mpi1`: `docker build -t mpi1 .`

```
(base) root@RAINBOW:                                    # docker build -t mpi1 .
[+] Building 1.2s (10/10) FINISHED                                                  docker:default
 => [internal] load build definition from Dockerfile                                          0.0s
 => => transferring dockerfile: 1.33kB                                                        0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest                              1.1s
 => [internal] load .dockerignore                                                             0.0s
 => => transferring context: 2B                                                               0.0s
 => [1/6] FROM docker.io/library/ubuntu:latest@sha256:dfc10878be8d8fc9c61cbff33166cb1d1fe44391539243703c72f8894ed7ef3e                0.0s
 => CACHED [2/6] RUN apt-get update &&     apt-get install -y     build-essential     openssh-server     openmpi-bin     openmpi-comm  0.0s
 => CACHED [3/6] RUN echo 'root:root' | chpasswd                                              0.0s
 => CACHED [4/6] RUN useradd -rm -d /home/mpi -s /bin/bash -g root -G sudo -u 1001 mpi &&     echo 'mpi:mpi' | chpasswd              0.0s
 => CACHED [5/6] RUN mkdir -p /home/mpi/.ssh &&     ssh-keygen -q -t rsa -N '' -f /home/mpi/.ssh/id_rsa &&     cat /home/mpi/.ssh/id_  0.0s
 => CACHED [6/6] WORKDIR /home/mpi                                                            0.0s
 => exporting to image                                                                        0.0s
 => => exporting layers                                                                       0.0s
 => => writing image sha256:20cc18cc1d321da7a4c696f49cf29ad0ccc1e3f07b33ae7889313f9e2f9c5f5d  0.0s
 => => naming to docker.io/library/mpi1                                                       0.0s
```

TASK: Test MPI with Docker Compose

Download the sample code zip file from Blackboard and try to run yourself. This sample executes the MPI Hello World program we wrote previously on 2 containers via Docker Compose. Follow the following steps:

1. Build an image named `mpi1`: `docker build -t mpi1 .` ←**The dot is important!**
2. Start 2 containers: `docker compose up -d`
3. Access the first container: `docker exec -it node1 /bin/bash`
4. Switch to the 'mpi' user: `su mpi`
5. Write host file for `mpirun`:
   a. `echo node1 slots=2 > host`
   b. `echo node2 slots=2 >> host`
6. Grant execution permission to the MPI Hello World program (password=`mpi`): `sudo chmod 777 ./mpi_files/intro`
7. Execute the MPI Hello World program: `mpirun -n 4 --hostfile host ./mpi_files/intro`

```
mpi@node1:~$ mpirun -n 4 --hostfile host ./mpi_files/intro
Hello world from processor node1, rank 0 out of 4 processors
Hello world from processor node1, rank 1 out of 4 processors
Hello world from processor node2, rank 3 out of 4 processors
Hello world from processor node2, rank 2 out of 4 processors
```

```
(base) root@RAINBOW:/home/rainbow/aixlab/docker_dist_mpi# docker compose up -d
[+] Running 3/3
 ✔ Network docker_dist_mpi_mpi-network  Created
 ✔ Container node2                      Started
 ✔ Container node1                      Started
```

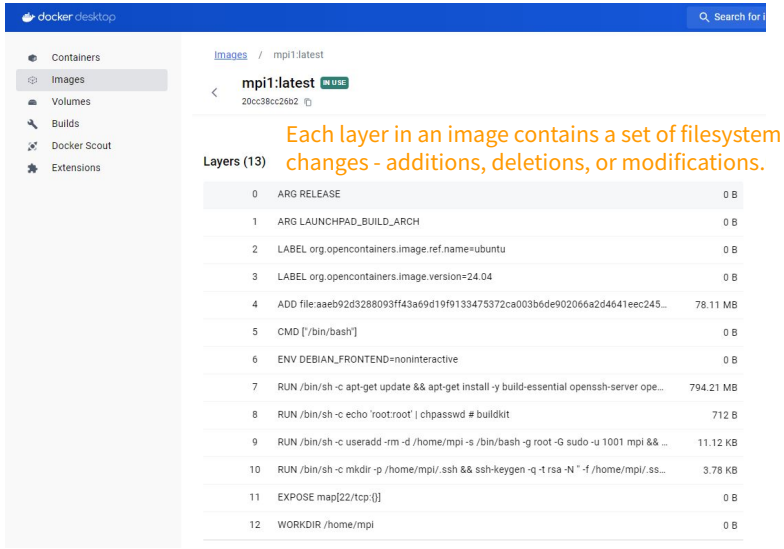
```
(base) root@RAINBOW:/home/rainbow/aixlab/docker_dist_mpi# docker exec -it node1 /bin/bash
root@node1:/home/mpi# su mpi
mpi@node1:~$ echo node1 slots=2 > host
mpi@node1:~$ echo node2 slots=2 >> host
mpi@node1:~$ cat host
node1 slots=2
node2 slots=2
```

```
mpi@node1:~$ sudo chmod 777 ./mpi_files/intro
[sudo] password for mpi:
```

TASK: Test MPI with Docker Compose

`Dockerfile`:

- **`FROM`**: define a base image.
- **`ENV`**: set environment variables.
- **`RUN`**: execute build commands.
- **`EXPOSE`**: describe which ports you application is listening on.
- **`WORKDIR`**: change working directory.



Dockerfile

```dockerfile
# Use the official Ubuntu base image
FROM ubuntu

# Set environment variables to prevent interactive prompts during package installations
ENV DEBIAN_FRONTEND=noninteractive

# Update the package list and install necessary packages
RUN apt-get update && \
    apt-get install -y \
    build-essential \
    openssh-server \
    openmpi-bin \
    openmpi-common \
    libopenmpi-dev \
    nano \
    sudo && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# set roots passw to root
RUN echo 'root:root' | chpasswd

# Create the mpi user
RUN useradd -rm -d /home/mpi -s /bin/bash -g root -G sudo -u 1001 mpi && \
    echo 'mpi:mpi' | chpasswd

# Set up SSH configuration for the mpi user
RUN mkdir -p /home/mpi/.ssh && \
    ssh-keygen -q -t rsa -N '' -f /home/mpi/.ssh/id_rsa && \
    cat /home/mpi/.ssh/id_rsa.pub >> /home/mpi/.ssh/authorized_keys && \
    # Set ownership to mpi user and correct permissions
    chown -R mpi:root /home/mpi/.ssh && \
    chmod 700 /home/mpi/.ssh && \
    chmod 600 /home/mpi/.ssh/authorized_keys && \
    echo "StrictHostKeyChecking no" >> /home/mpi/.ssh/config

# Expose the SSH port
EXPOSE 22

# Set the working directory to ...
WORKDIR /home/mpi
```

TASK: Test MPI with Docker Compose

`compose.yaml`:

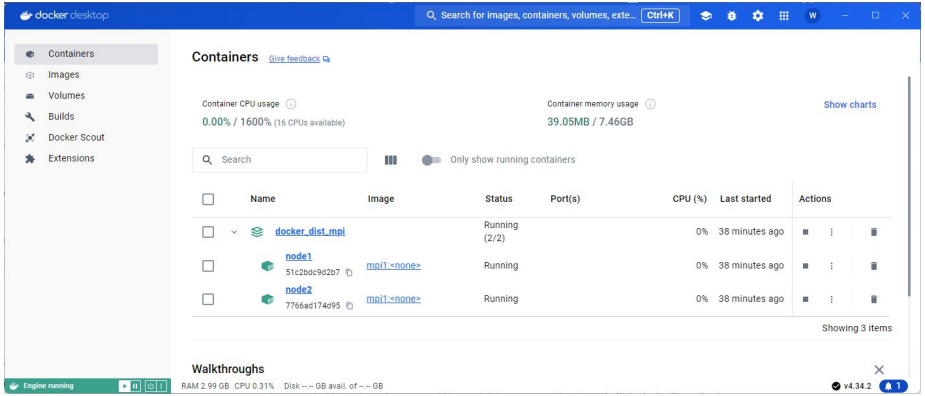
- **`container_name`**: used for accessing container by name.
- **`hostname`**: used for mpirun to recognize hosts by host file.
- **`image`**: specify the mpi1 image we built.
- **`networks`**: configure a basic bridge network between 2 hosts.
- **`command`**: override the default command by the container image.
- **`Volumes`**: define mount host paths or named volumes that are accessible by service containers.

compose.yaml

```yaml
services:
  node1:
    container_name: node1
    hostname: node1
    image: mpi1
    networks:
      - mpi-network
    # start the ssh service so that other containers can connect to us && wait forever
    command: /bin/bash -c "
      service ssh start &&
      tail -f /dev/null"
    # allow the container to access the mpi_files directory on the host system from the container directory "/home/mpi/mpi_files/"
    volumes:
      - ./mpi_files:/home/mpi/mpi_files/

  node2:
    container_name: node2
    hostname: node2
    image: mpi1
    networks:
      - mpi-network
    # start the ssh service so that other containers can connect to us && wait forever
    command: /bin/bash -c "
      service ssh start &&
      tail -f /dev/null"
    # allow the container to access the mpi_files directory on the host system from the container directory "/home/mpi/mpi_files/"
    volumes:
      - ./mpi_files:/home/mpi/mpi_files/

networks:
  mpi-network:
    driver: bridge
```



TASK: Test MPI with Docker Compose

Download the sample code zip file from Blackboard and try to run yourself. This sample executes the MPI Hello World program we wrote previously on 2 containers via Docker Compose.

As a final step, clean up the resources by: `docker compose down`

That's the end for this task! If you want to play with Docker Compose a bit more, refer to this link: https://docs.docker.com/compose/gettingstarted/.