Share same OS kernel
↑
Containers — have there own → Processes, Network, Mounts

* Way to package application with all necessary dependencies & configuration
* Portable artifact, easily shared & moved around
* Makes development & deployment more efficient.

Where do Containers live?
- Container Repository
- Many comp have → Private Repository
- Docker hub → Public repo for docker

Application development before containers

Installation process different on each OS environment

Many steps, something could go wrong

PostgresSQL
Redis

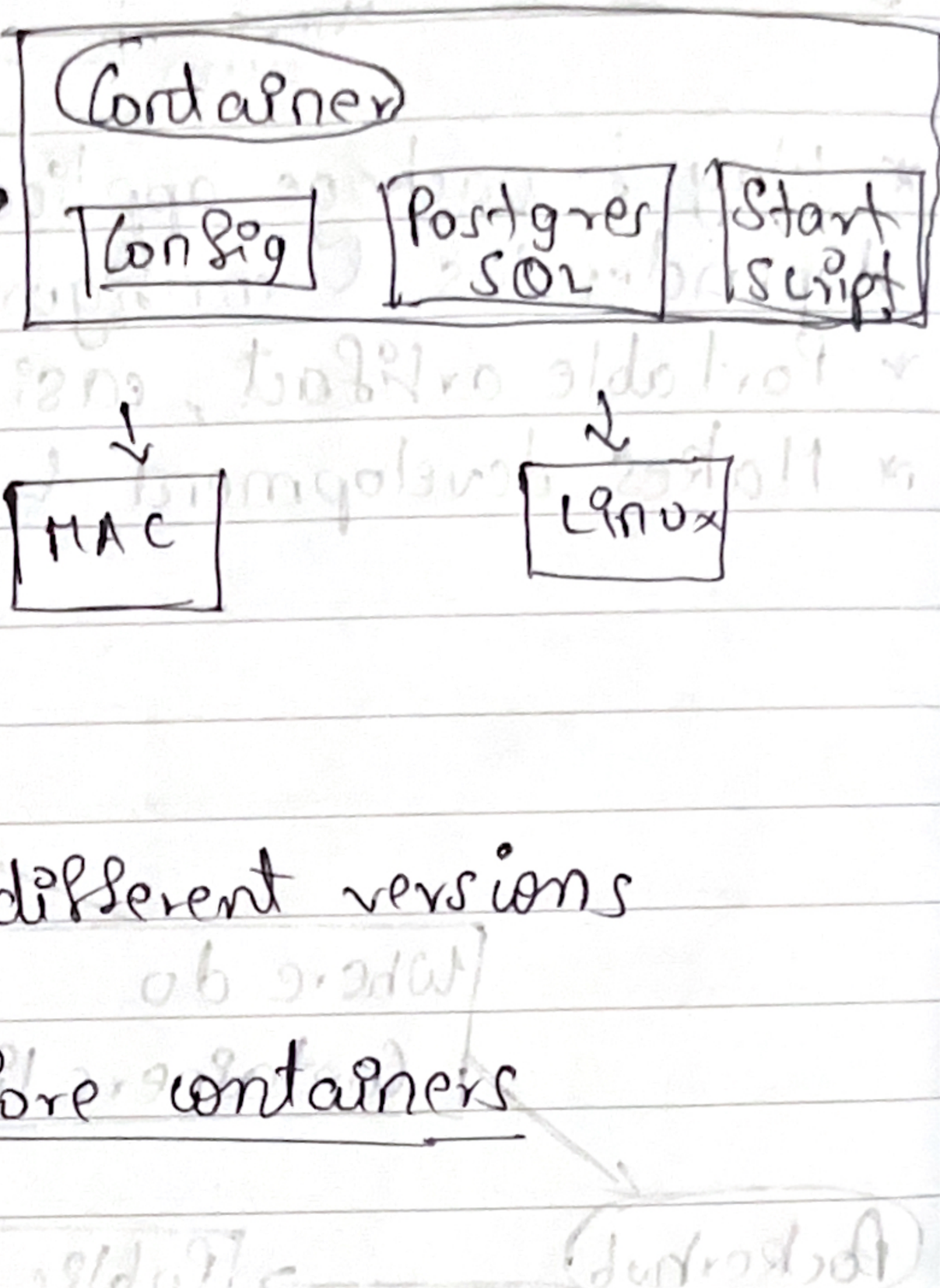
PostgresSQL
Redis

MAC — Developer 1

Linux — developer 2

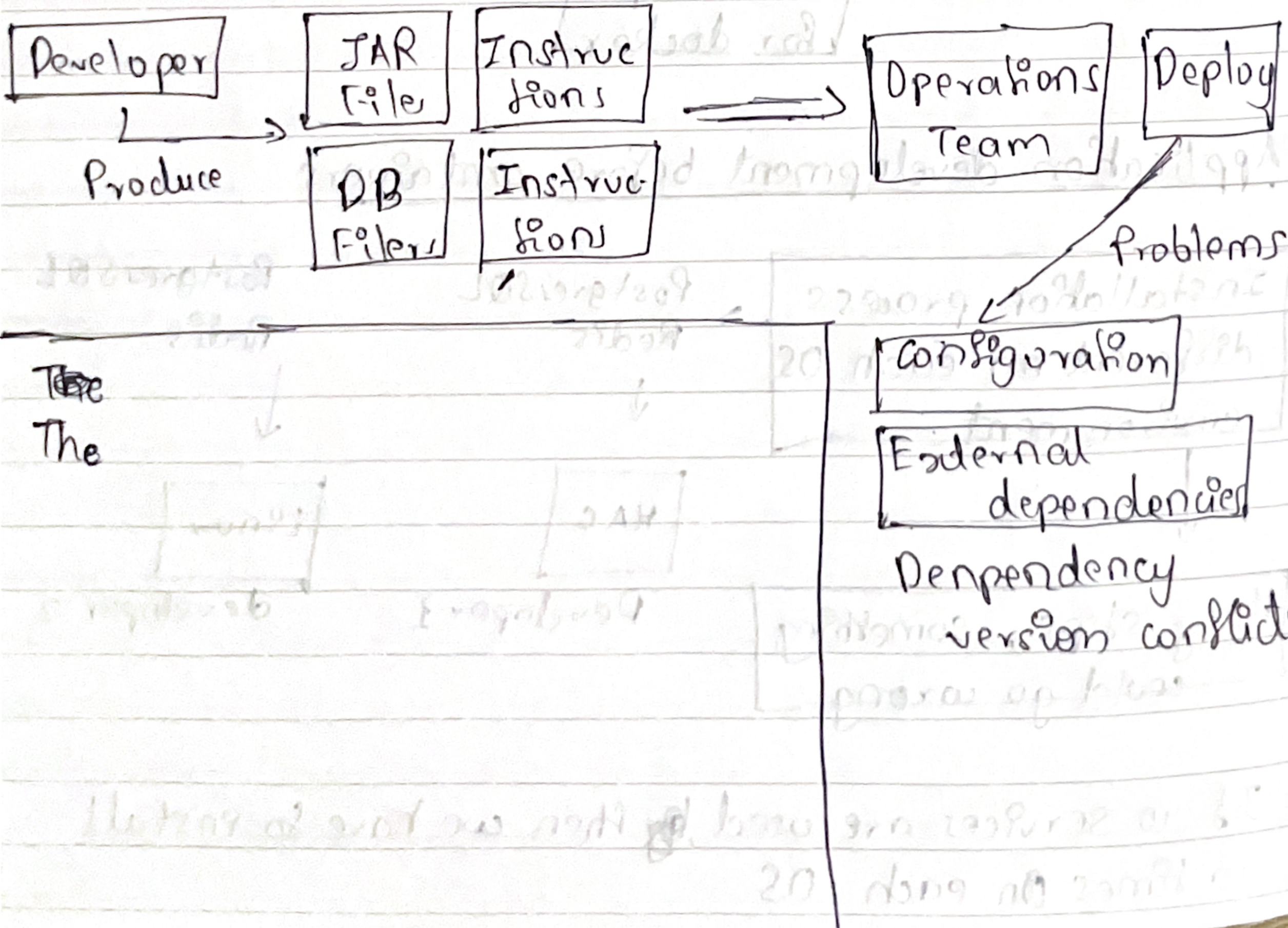
* If 10 services are used, then we have to install 10 times on each OS

Application development after containers



* Isolated environment
* Packaged with all needed configuration
* One command to install the app
* Run same app with 2 different versions

Application deployment before containers

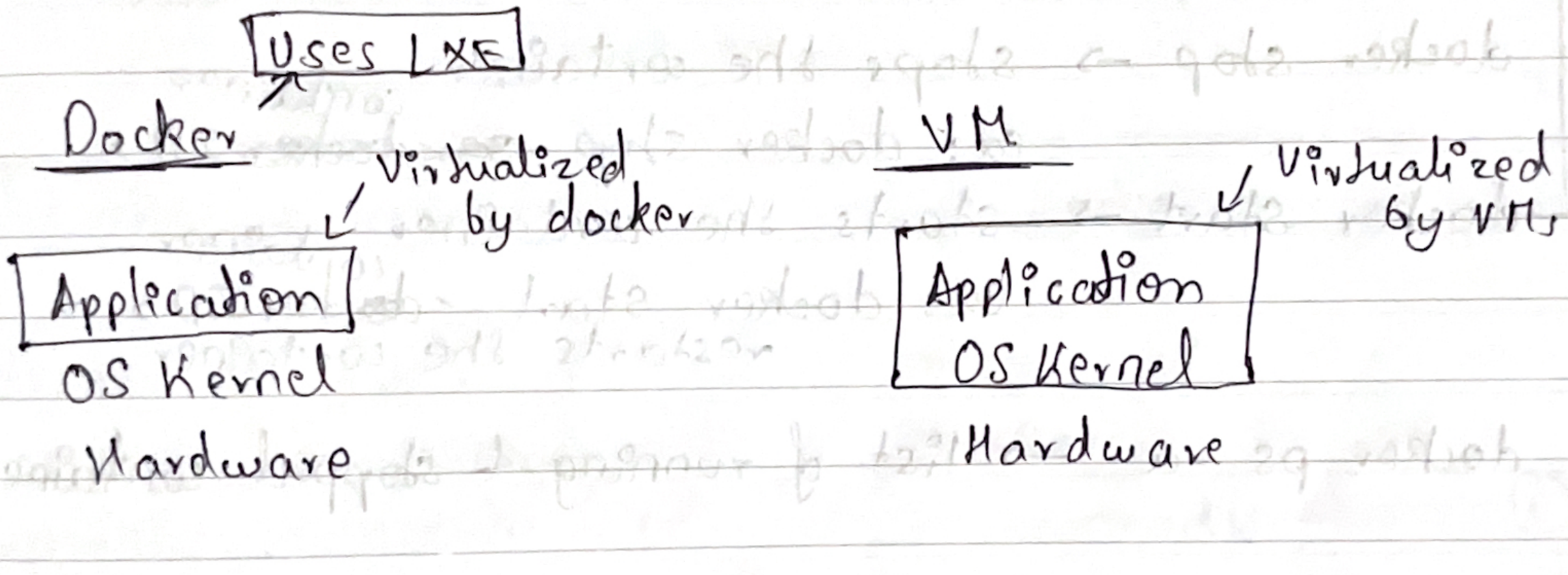


The

The

Application deployment after containers

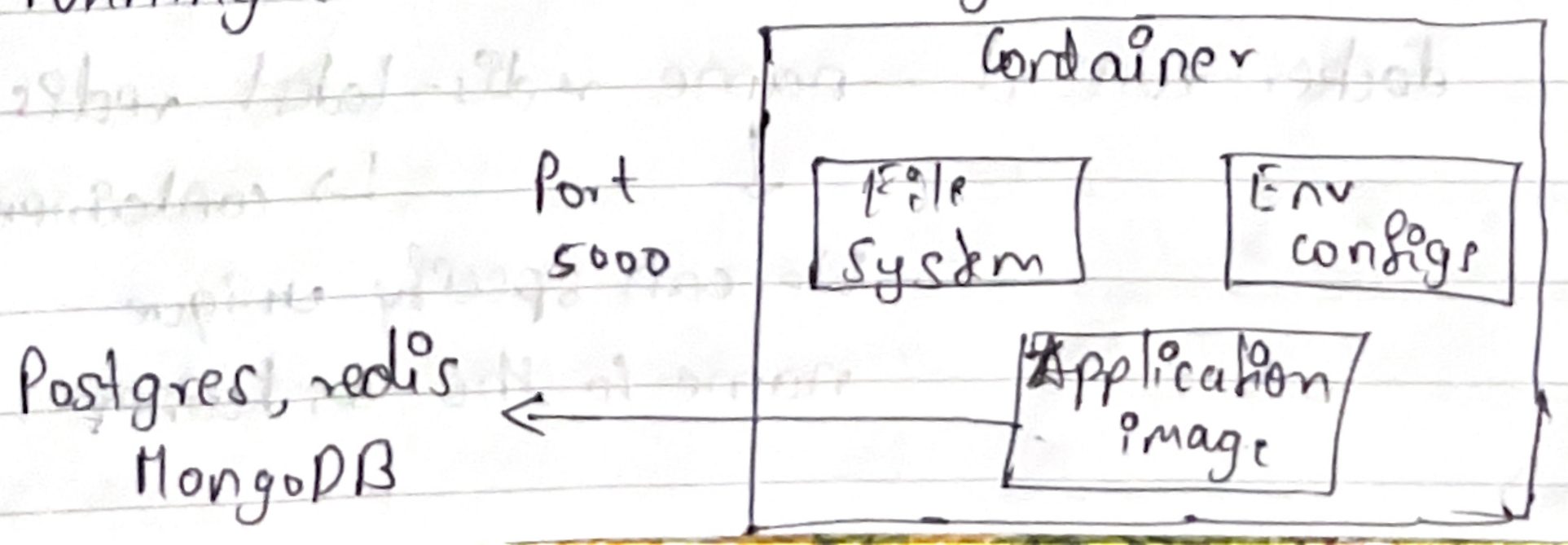
* Developers & operations work together to package the application in container
* No environmental configuration needed on server (except docker runtime)



* Size: Small — Large
* Speed: Faster — Slower compared to docker
* Compatability: cannot be done in docker — can run Virtual OS on on any Host OS

Difference between image & container

* Container is running environment for image



Docker commands

docker ps → list running containers

docker run → also pulls image & starts container
→ starts new container with command
ex: docker run -d redis

docker stop → stops the container
ex: docker stop <container ID>

docker start → starts the container
ex: docker start <container ID>
restarts the container

docker ps -a → list of running & stopped container

* We can run two different versions of images by specifying ports (port binding)
ex: docker run -p6000:6379 redis:4.0
docker run -p6001:6379 redis

Host OS Port ← 6001 : 6379 → Container port

· docker logs <container ID> → prints logs

docker run -d --name redis-latest redis
↓ ↳ container name given by user
We can specify unique name to the container

docker exec -it <containerID> /bin/bash → used to get container terminal

↓

interactive terminal

↳ We can also specify name given by user

* Docker compose → for running multiple docker containers

docker command	mongo-docker-compose.yaml

```
docker run -d \                          version: '3'
--name mongodB \ (1)                     services:
-p 27017:27017 \ (3)                       mongodB (1)
-e <environment variables>                   image: mongo (2)
(4)                                          ports:
-e "                                           -27017:27017
--net mongo-network \                        environment: (3)
mongo (2)                                    - (4)
```

① container name
② image
③ ports
④ environment variables

Note:
Indentation in yaml file should be correct

docker-compose -f mongo.yaml up/down

↓

Creates default network

↳ which will start all the containers which are in mongo.yaml file

Shuts down all the containers

* To deploy an application it should be packaged into its own docker container
  ↳ We should build docker image

Dockerfile : Blueprint for building images

Image environment Blueprint	Docker File
install node	FROM node (1)
set MONGO_DB_USERNAME= PASSWORD=	ENV MONGO_DB_USERNAME= PASSWORD= (2)
create /homelapp folder	RUN mkdir -p /homelapp (3)
copy current folder files to /homelapp	COPY . /homelapp (4)
start the app : node server.js	CMD ["node", "server.js"] (5)

① Start by basing it on another image
② Optionally define environment variables
③ RUN → execute any Linux command
   ↳ /home/app directory is created inside container [not on laptop, not on host] / can have multiple RUN
④ COPY command executes on Host machine
⑤ Executes the entry point linux command

Building docker image

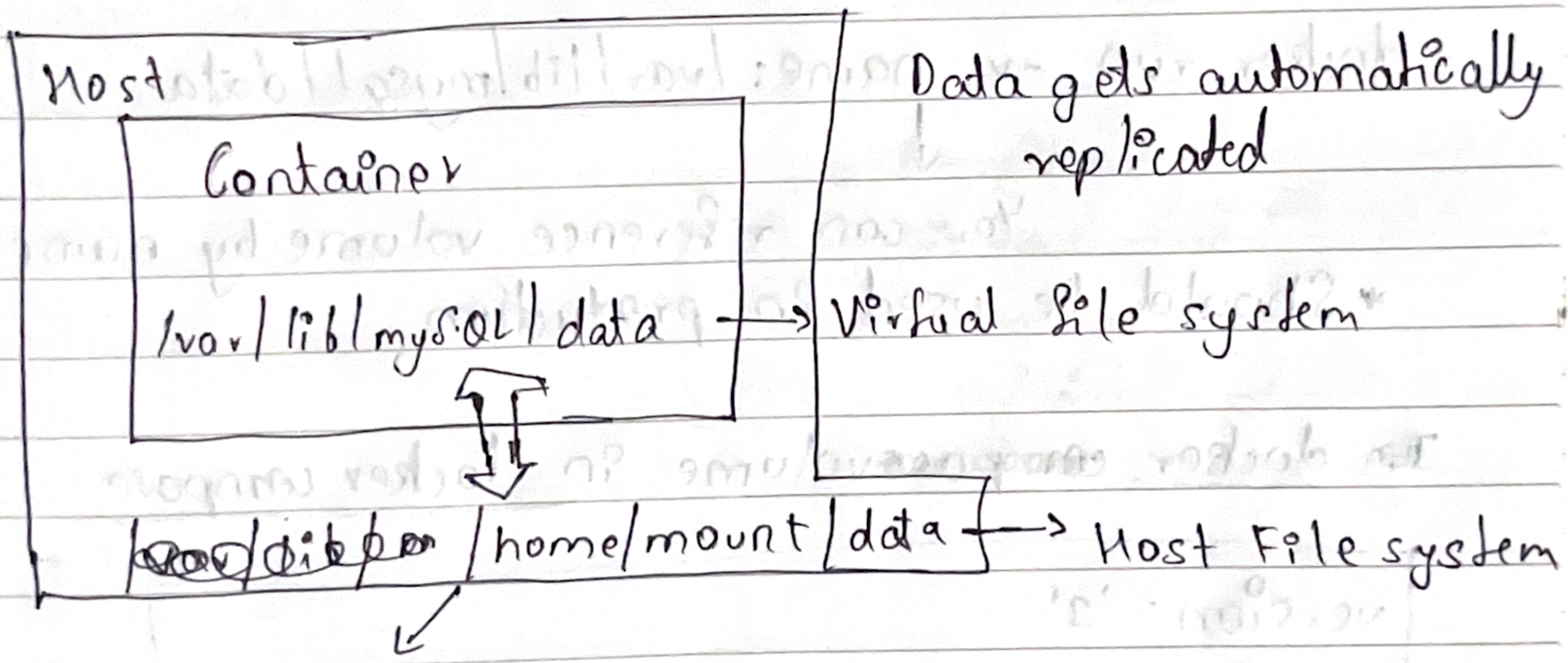
Tag

```
docker build -t my-app:1.0 <dockerFile>
```

↳ Image name

Docker Volumes

These are used for data persistence



Folder in physical host file system is mounted into virtual file system for docker

3 Volume Types

↙ Host Volumes

```
docker run -v /home/mount/data:/var/lib/mysql/data
```

↙ advantages

You decide where on host file system the reference is made

↓ Host directory

↓ Container directory

ii) Anonymous volumes

docker run -v /var/lib/mysql/data

↳ For each container a folder is generated that gets mounted (automatically created by docker)

iii) Named volumes

docker run -v name: /var/lib/mysql/data

↓

You can reference volume by name

* Should be used in production

docker compose volume in docker compose

```
version: '3'
services:
    mongodb:
        image: mongo
        ports:
            - 27017:27017
        volumes:
            - db-data: /var/lib/mysql/data
```

→ .yaml file

↓

Named volume