Week7:

Virtualization

Shahid Beheshti University (SBU)
Alireza Shameli

http://cloudsec.sbu.ac.ir/

Content

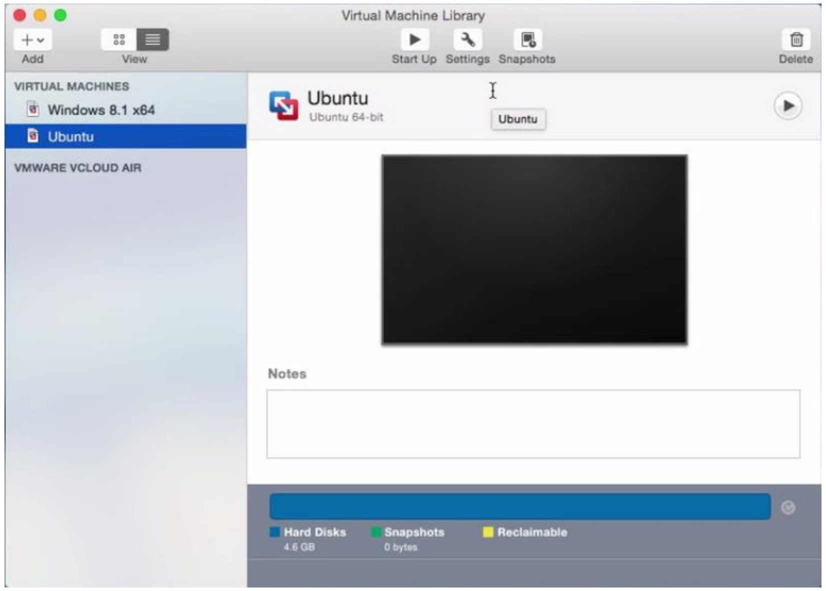
- Definition of Virtualization
- Server Virtualization
- Definition of a Hypervisor
- Type 1 vs. Type 2 Hypervisors
- Pros and Cons of Server Virtualization
- Pros and Cons of dedicated Server
- Containers

What is virtualization?

- Virtualization: extend or replace an existing interface to mimic the behavior of another system.
  - Introduced in 1970s: run legacy software on newer mainframe hardware
- Handle platform diversity by running apps in VMs
  - Portability and flexibility

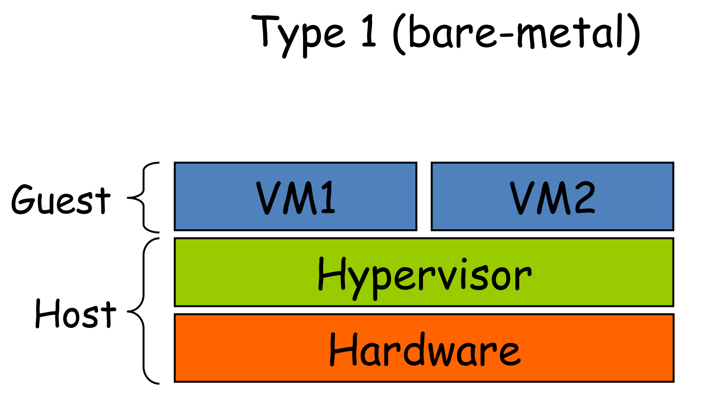
What is virtualization?

- Virtualization is way to run **multiple operating systems** and **user applications** on the same hardware
  - E.g., run both Windows and Linux on the same laptop
- How is it different from **dual-boot**?
  - Both OSes run **simultaneously**
- The OSes are completely **isolated** from each other

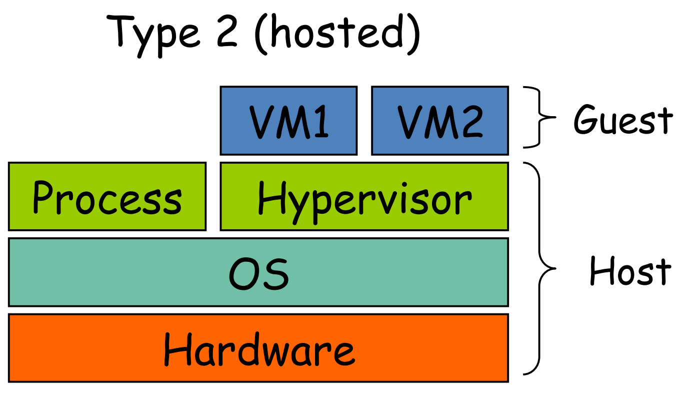


Two Types of Hypervisors

- Definitions
  - **Hypervisor** (or **VMM** – Virtual Machine Monitor) is a software layer that allows several **virtual machines** to **run** on a **physical machine**
  - The physical OS and hardware are called the **Host**
  - The virtual machine OS and applications are called the **Guest**



VMware ESX, Microsoft Hyper-V, Xen



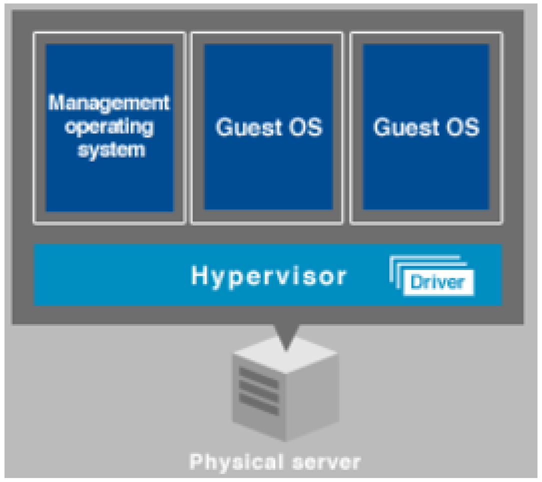
VMware Workstation, Microsoft Virtual PC, Sun VirtualBox, QEMU, KVM

Example of Hypervisors

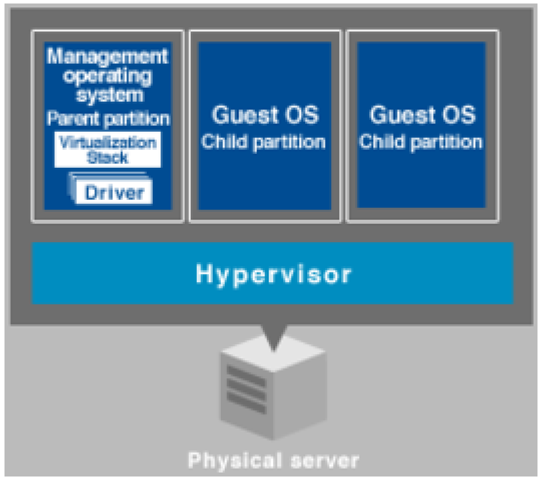
- Type 1 Hypervisor (loaded directly on the hardware)
  - VMware ESX/ESXi
  - Hyper-V
  - XenServer
- Type 2 Hypervisor (loaded in an OS running on the hardware)
  - Fusion
  - Virtual Server
  - VMware Workstation

Other Categories

- Monolithic Hypervisor
  - Contains its own drivers model
- Microkernel Hypervisor
  - Drivers run within guests



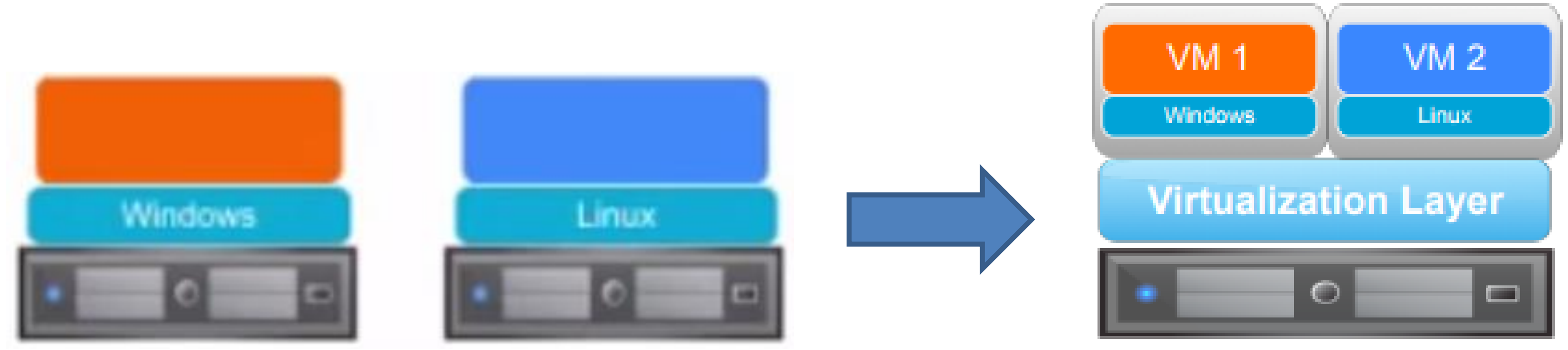
Monolithic



Microkernel

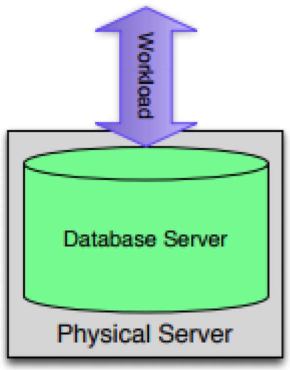
Server Virtualization

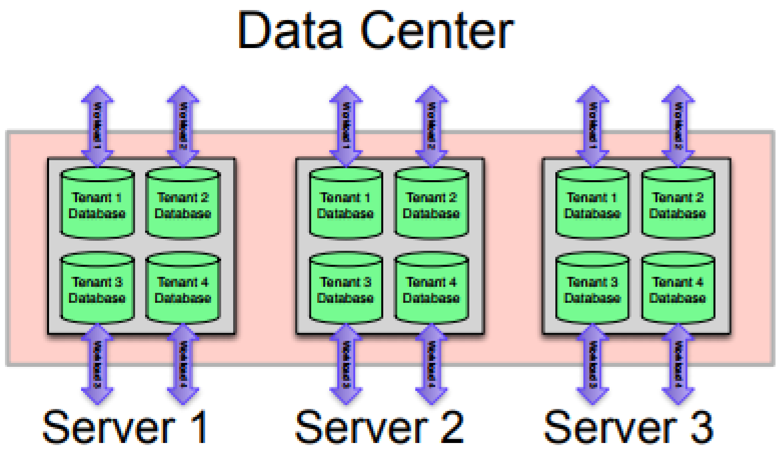
- Allows a server to be “sliced” into Virtual Machines (VMs)
- VM has own OS/applications
- Rapidly adjust resource allocation



Example: Virtualized DB Servers

- Conventional: one physical server, one database server
- Data center: multiple physical servers, multiple database servers per (virtualized) physical server





Pros and Cons of Server Virtualization

- Pros
  - Cost
    - Less physical servers
    - Less server space (consolidation of servers)
    - Less energy costs
    - Less maintenance
  - Efficient Administration
    - Easier management, management through one machine
    - Smaller IT staff

Pros and Cons of Server Virtualization

- Pros
  - Growth and Scalability
    - Upgrading one server upgrades them all
    - Easy growth
  - Security
    - Single server security maintenance
    - Hypervisor software often provides security benefits

Pros and Cons of Server Virtualization

- Cons
  - Slow Performance
    - High stress on single machine
    - Longer processing times
    - More network bottlenecking
  - Single Point of Failure
    - Many servers on one host machine
    - Hardware or software failures can be critical
    - Backup servers will need to be setup

Pros and Cons of Server Virtualization

- Cons
  - Cost
    - High initial investment
    - Software licensing costs
  - Security
    - All servers through one machine
  - Learning curve
    - Many different types of software
    - Different architecture

Pros and Cons of Dedicated Servers

- Pros
  - High Performance
    - All resources on server are dedicated
    - Can handle high stress scenarios
  - Multiple Points of Failure
    - Easier to identify problems
    - Only one server will fail at a time

Pros and Cons of Dedicated Servers

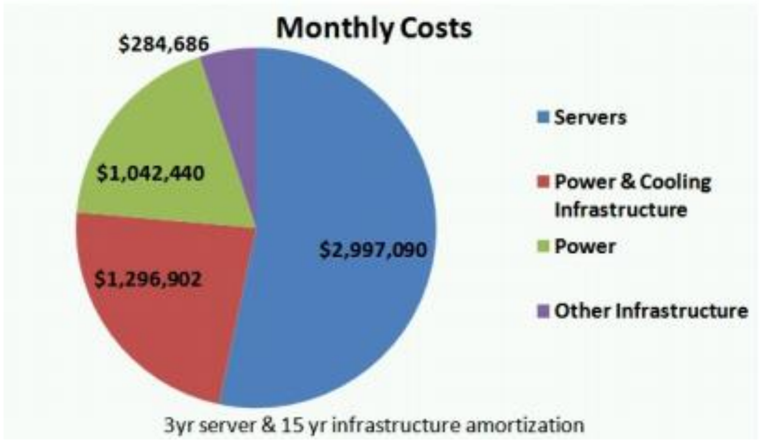
- Pros
  - Price
    - Old servers already exist
    - No long term investments
    - If it’s not broke, don’t fix it
  - Small Learning Curve
    - Dedicated servers have been around for a long time
    - IT staff will not need to learn any new systems if dedicated servers already exist

Pros and Cons of Dedicated Servers

- Cons
  - Price
    - Long term costs of dedicated servers can add up
    - More applications and services = more servers
  - Servers not being utilized
    - Servers may not be efficient
    - Even at peak, some servers may not need all resources

Data Center Costs

- Running a data center is expensive:

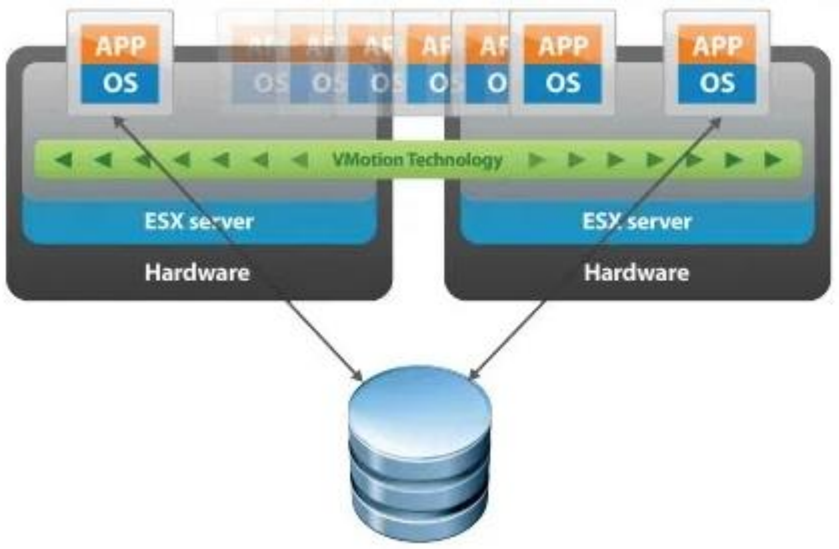


Amortized Cost	Component	Sub-Components
~45%	Servers	CPU, memory, storage systems
~25%	Infrastructure	Power distribution and cooling
~15%	Power draw	Electrical utility costs
~15%	Network	Links, transit, equipment

Guide to where costs go in the data center

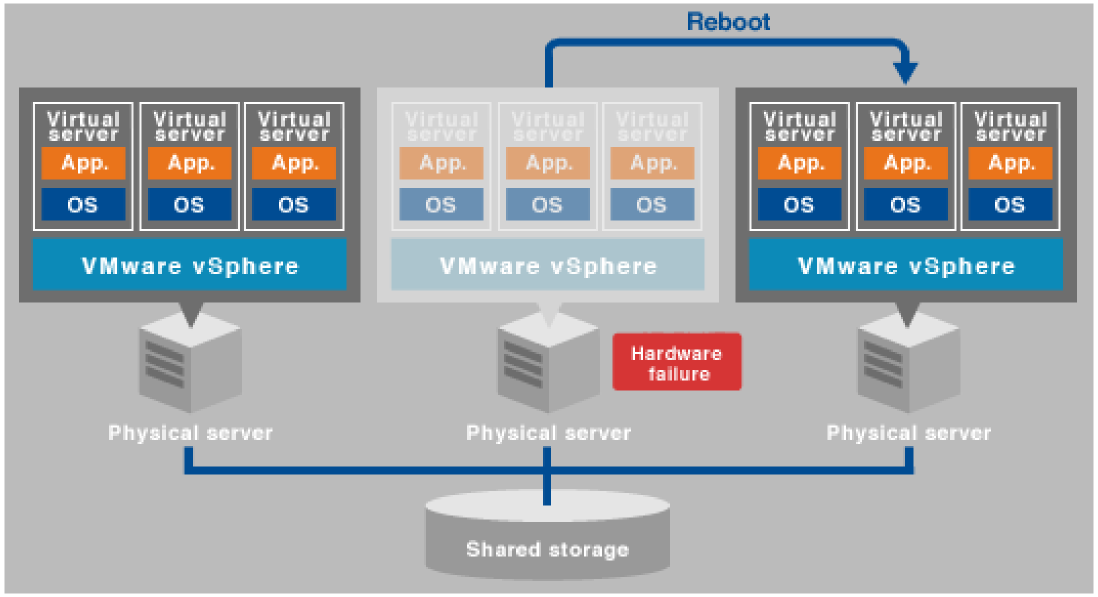
VM Live Migration

- Live migration of running virtual machines from one physical server to another are useful for:
  - Hardware maintenance
  - Performance and optimizes resource usage
  - Load balancing across physical
  - Disaster recovery



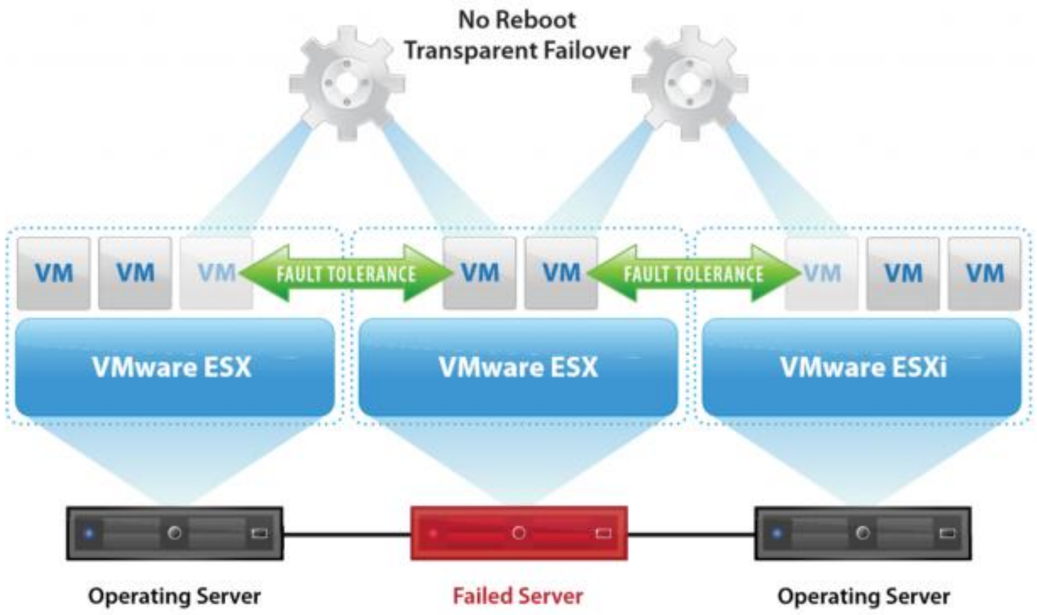
High Availability

- If a 15-second period elapses without the receipt of heartbeats from a host, and the host cannot be pinged, it is declared as failed. In the event of a host failure, the virtual machines running on that host are failed over, that is, restarted on alternate hosts.



Fault Tolerance

- Fault Tolerance provides a higher level of business continuity than VMware HA.
- When a Secondary VM is called upon to replace its Primary VM counterpart, the Secondary VM immediately takes over the Primary VM's role with the entire state of the virtual machine preserved.



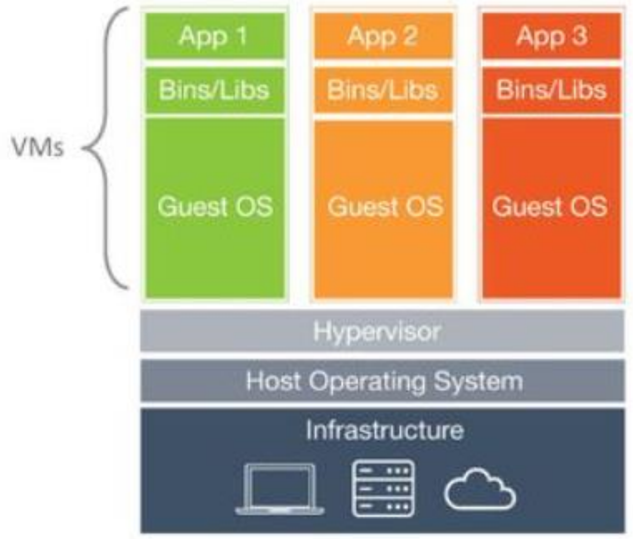
Different Types of Virtualization

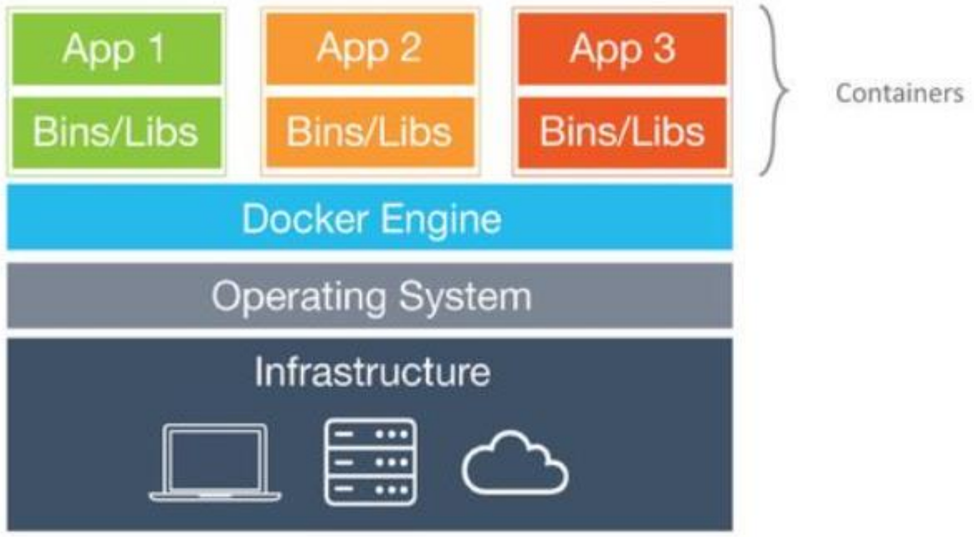
- Desktop Virtualization
- Application Virtualization
- Storage Virtualization
- Network Virtualization
- I/O Virtualization
- CPU Virtualization
- Memory Virtualization

Containers

- Containers are an abstraction at the app layer that packages code and dependencies together
- Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space
- Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems

Virtual Machines vs. Containers





Virtual Machines

- Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system**

Containers

- Containers include the app & all of its dependencies, but **share the kernel** with other containers.
- Run as an isolated process in userspace on the host OS
- Not tied to any specific infrastructure – containers run on any computer, infrastructure and cloud.

Pros and Cons of Containerizing Applications

- Running an Application on a Host Machine
  - To get the application working, you would additionally require other packages that your application depends upon. You might also want different versions of the same package running on your system
- Running an Application on a Virtual Machine
  - Because a VM is a separate entity, you don’t have the same issues of inflexibility that arise from running an application directly on hardware
- Running an Application on a Container
  - The container can hold the application and its dependencies it requires to run, the startup time, disk space consumption, and processing power is much lower than those of a VM

Challenges of Using Containers

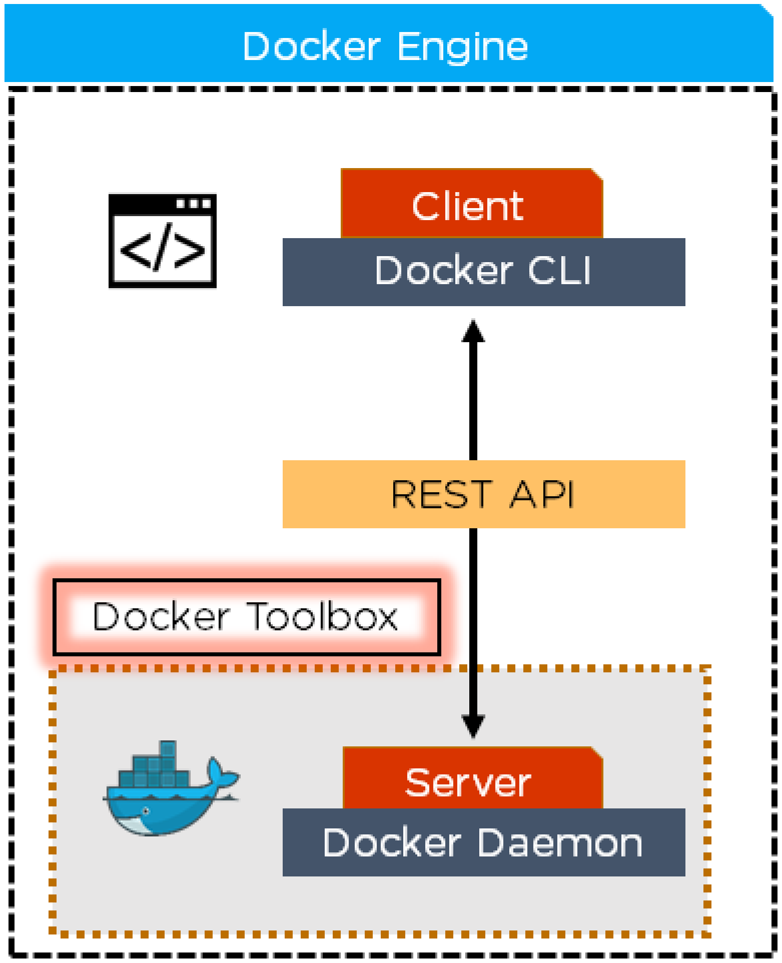
- Containers share the kernel and other components of the host OS. This means that containers are less isolated from each other than VMs, which have their own OS
- What happens when you want your container to work closely with another container

What is Docker?

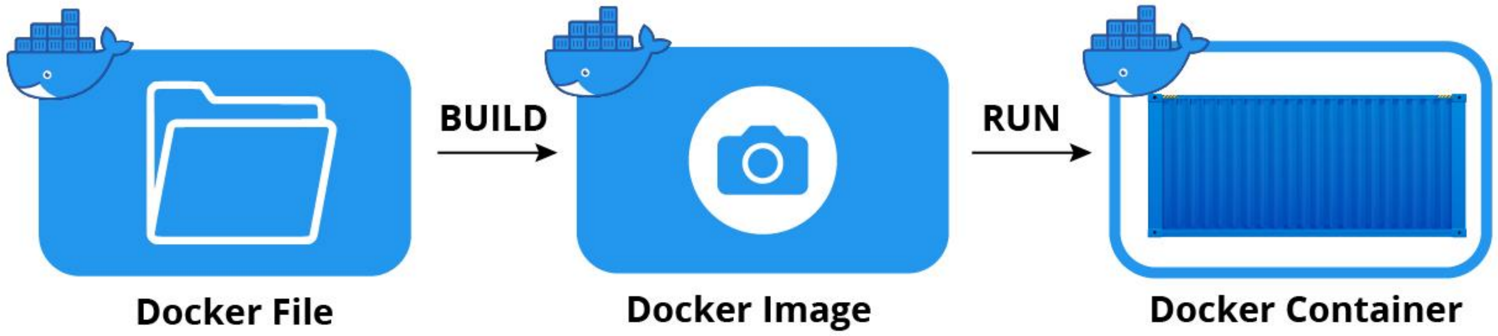
- Docker was released as an open source project by dotCloud, Inc., in 2013
- Docker enables developers to package applications into containers—standardized executable components that combine application source code with all the operating system (OS) libraries and dependencies required to run the code in any environment
- In order for Docker containers to scale, orchestration frameworks are key. In June 2014, Google introduced Kubernetes, which helped Docker scale. Later that year, Amazon’s EC2 container service, which is a cloud-based container as a service, was offered

Docker Engine

- The Docker engine is the infrastructure plumbing software that runs and orchestrates containers
- Docker Engine is the program that creates and runs the Docker container from the Docker image file
- It is made up of:
  - the Docker daemon, a server that is a type of long-running program; a REST API, which specifies interfaces that programs can use to talk to the daemon and tell it what to do; and
  - the CLI, the command-line interface that talks to the Docker daemon through the API.



Docker Image

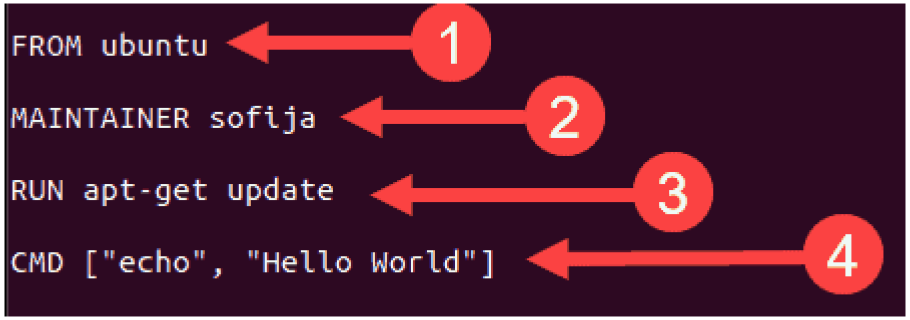


- A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform
- A **Dockerfile** is a script with instructions on how to build a Docker image. These instructions are, in fact, a group of commands executed automatically in the Docker environment to build a specific Docker image.

Example: “Hello World”

- Open the file with a text editor of your choice. In this example, we opened the file using Nano and add the following content:

```
nano Dockerfile
```



```
FROM ubuntu
MAINTAINER sofija
RUN apt-get update
CMD ["echo", "Hello World"]
```

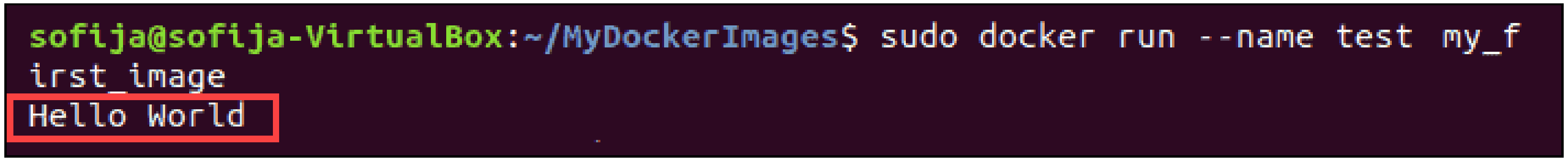
Example: “Hello World”

- Build a Docker Image with Dockerfile

```
docker build -t my_first_image Dockerfile
```

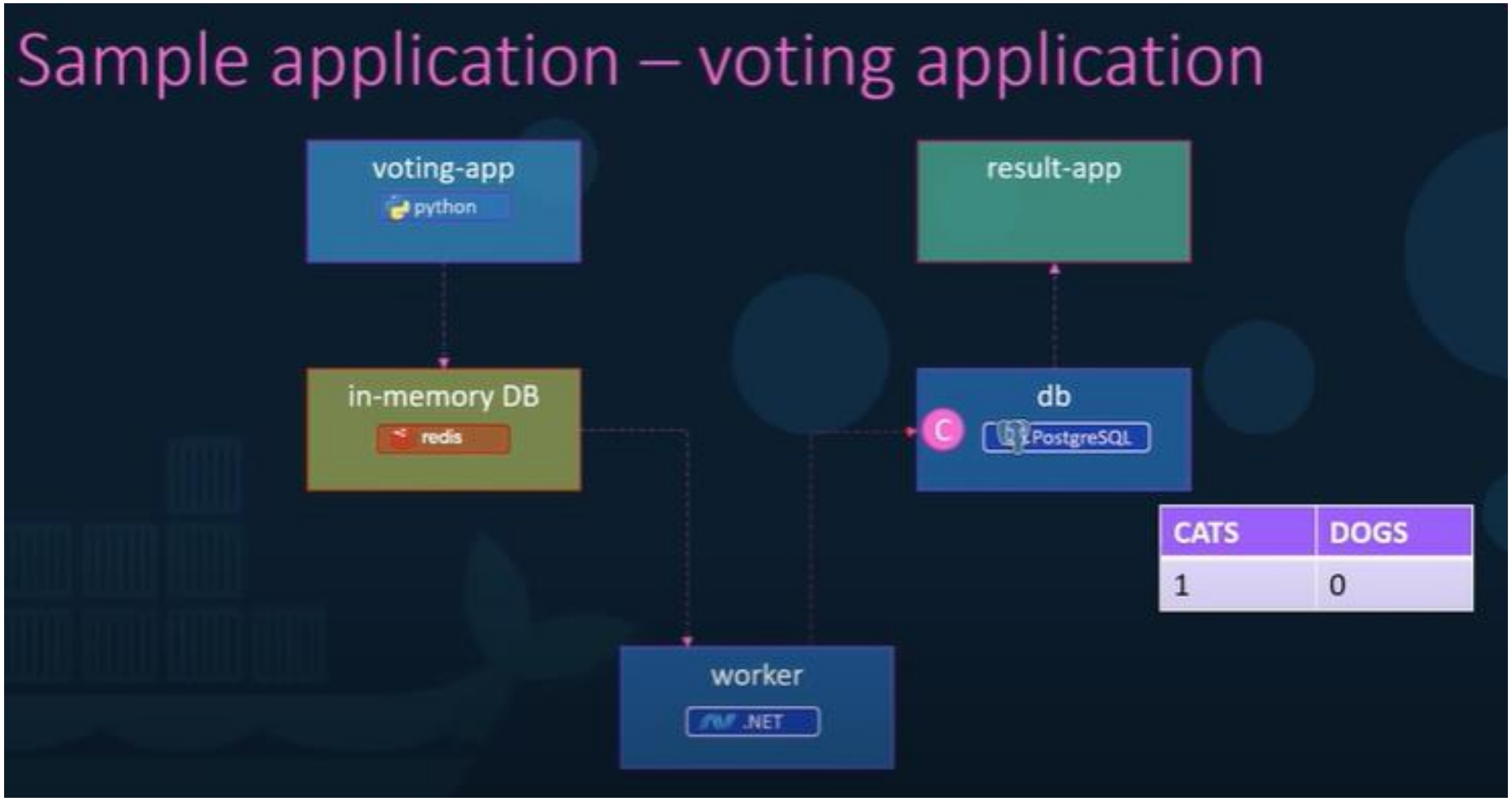
- Create a New Container

```
docker run --name test my_first_image
```



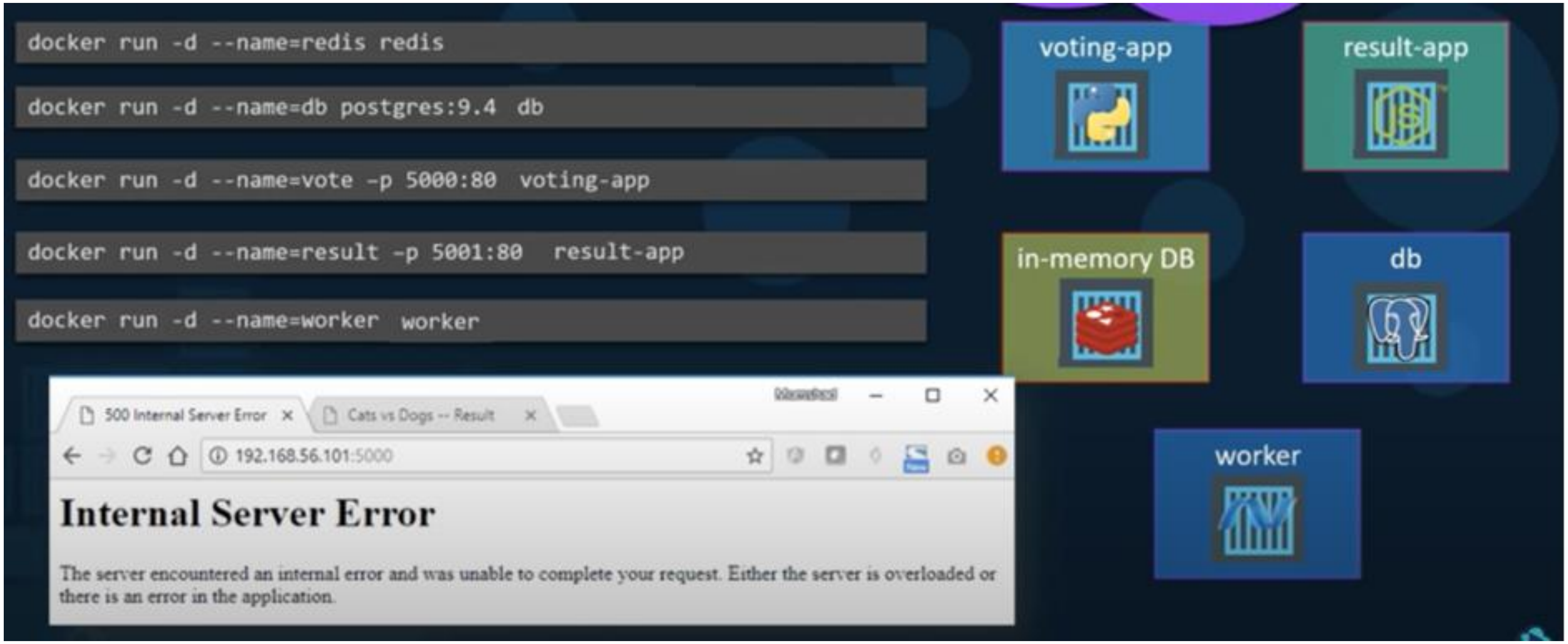
```
sofija@sofija-VirtualBox:~/MyDockerImages$ sudo docker run --name test my_f
irst_image
Hello World
```

Docker Compose



CATS	DOGS
1	0

Docker Compose

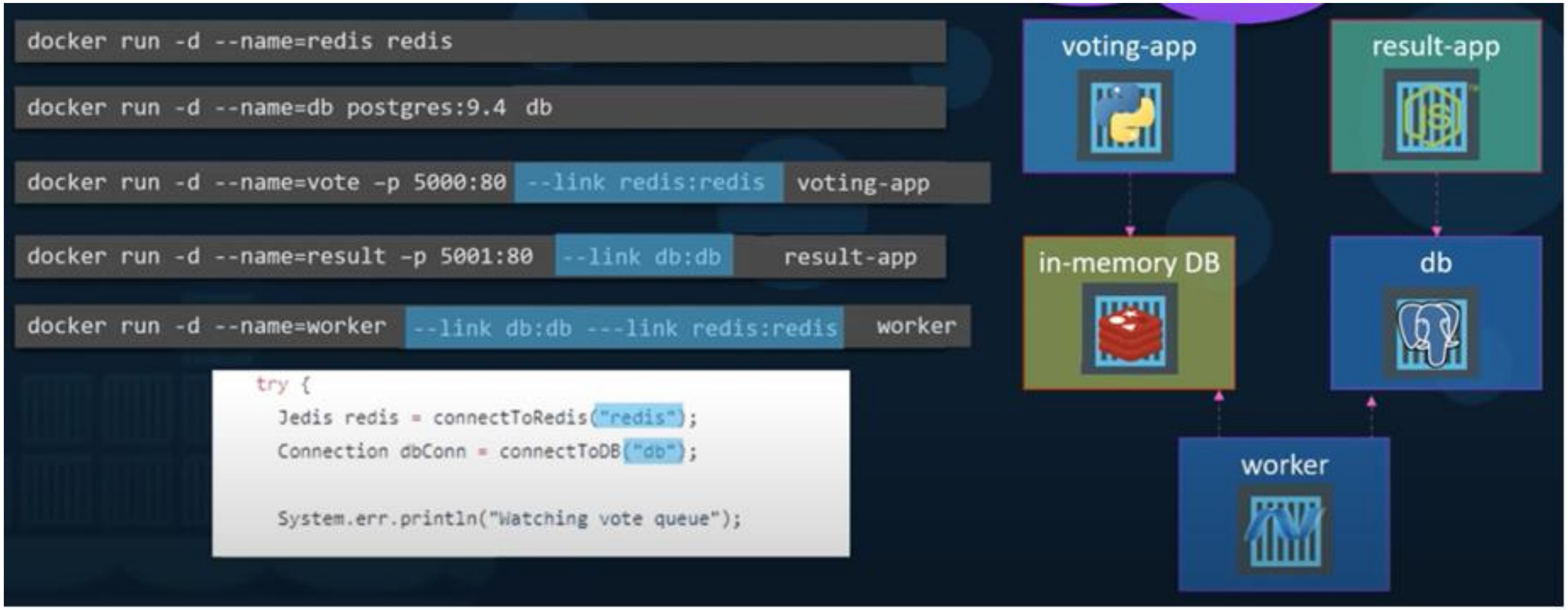


```
docker run -d --name=redis redis

docker run -d --name=db postgres:9.4  db

docker run -d --name=vote -p 5000:80  voting-app

docker run -d --name=result -p 5001:80   result-app

docker run -d --name=worker  worker
```

Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

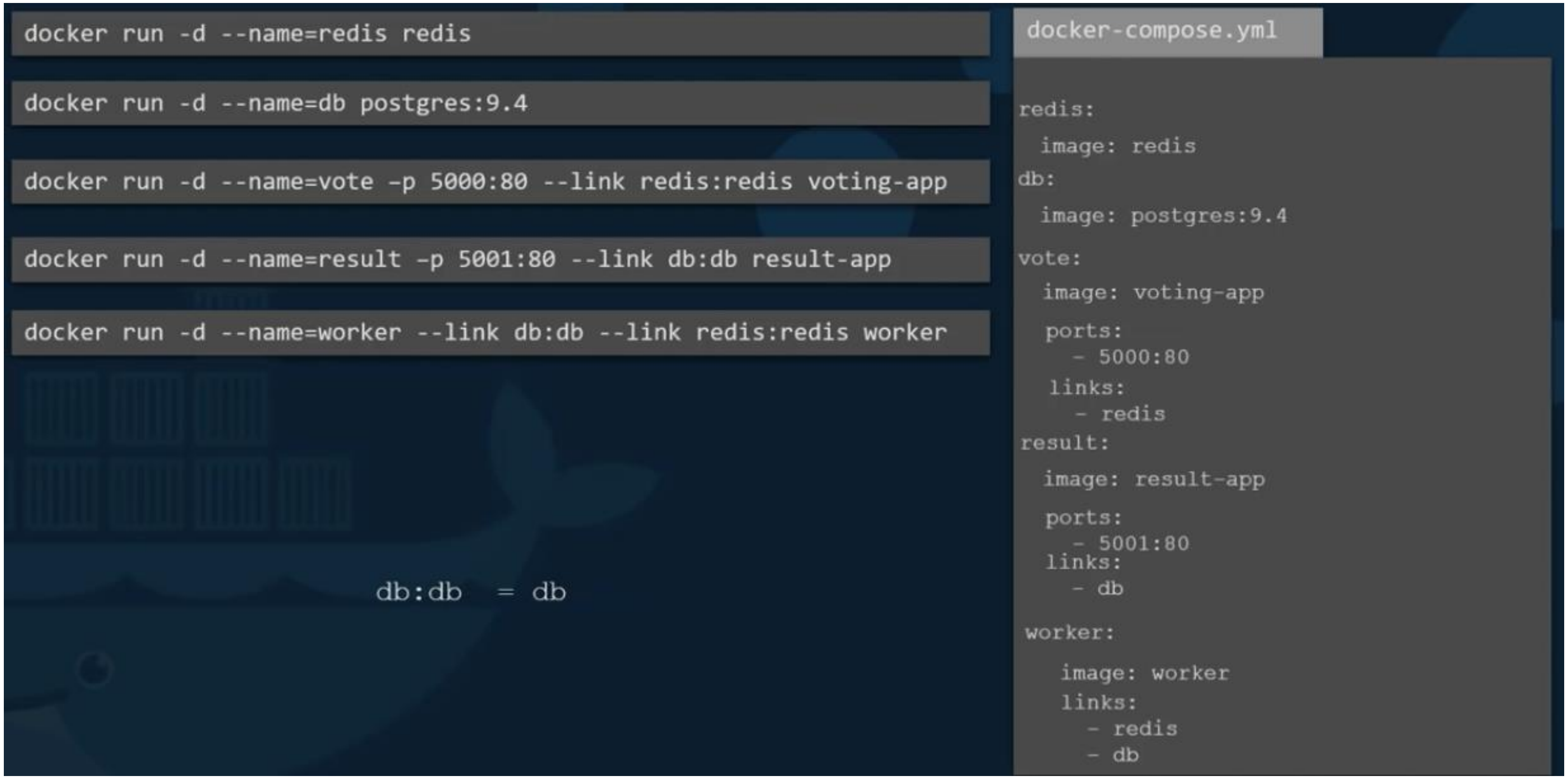
Docker Compose



```
docker run -d --name=redis redis

docker run -d --name=db postgres:9.4  db

docker run -d --name=vote -p 5000:80 --link redis:redis voting-app

docker run -d --name=result -p 5001:80 --link db:db  result-app

docker run -d --name=worker --link db:db ---link redis:redis  worker
```

```
try {
  Jedis redis = connectToRedis("redis");
  Connection dbConn = connectToDB("db");

  System.err.println("Watching vote queue");
```

Docker Compose



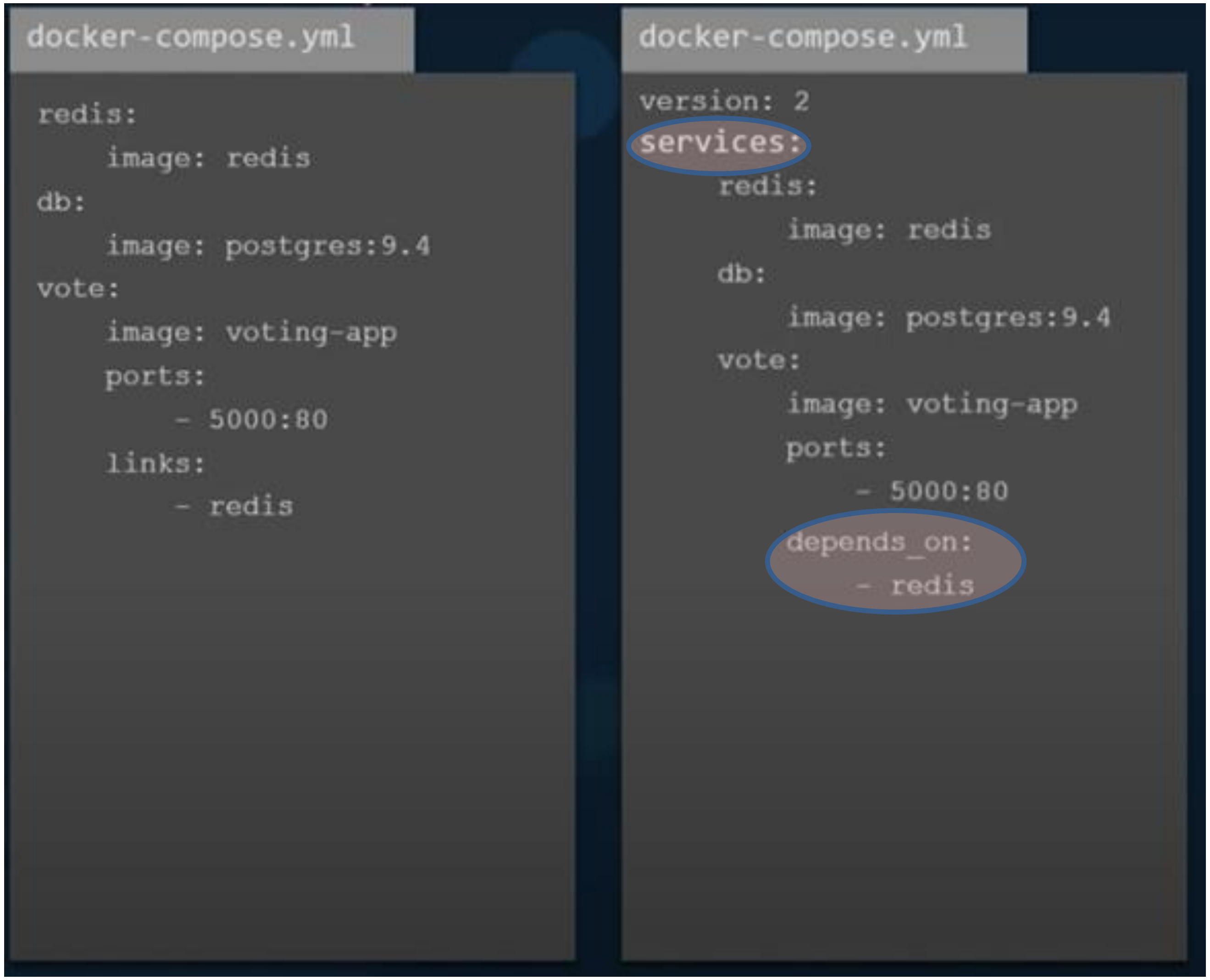
```
docker run -d --name=redis redis

docker run -d --name=db postgres:9.4

docker run -d --name=vote -p 5000:80 --link redis:redis voting-app

docker run -d --name=result -p 5001:80 --link db:db result-app

docker run -d --name=worker --link db:db --link redis:redis worker
```

db:db = db

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
vote:
  image: voting-app
  ports:
    - 5000:80
  links:
    - redis
result:
  image: result-app
  ports:
    - 5001:80
  links:
    - db
worker:
  image: worker
  links:
    - redis
    - db
```

Docker Compose



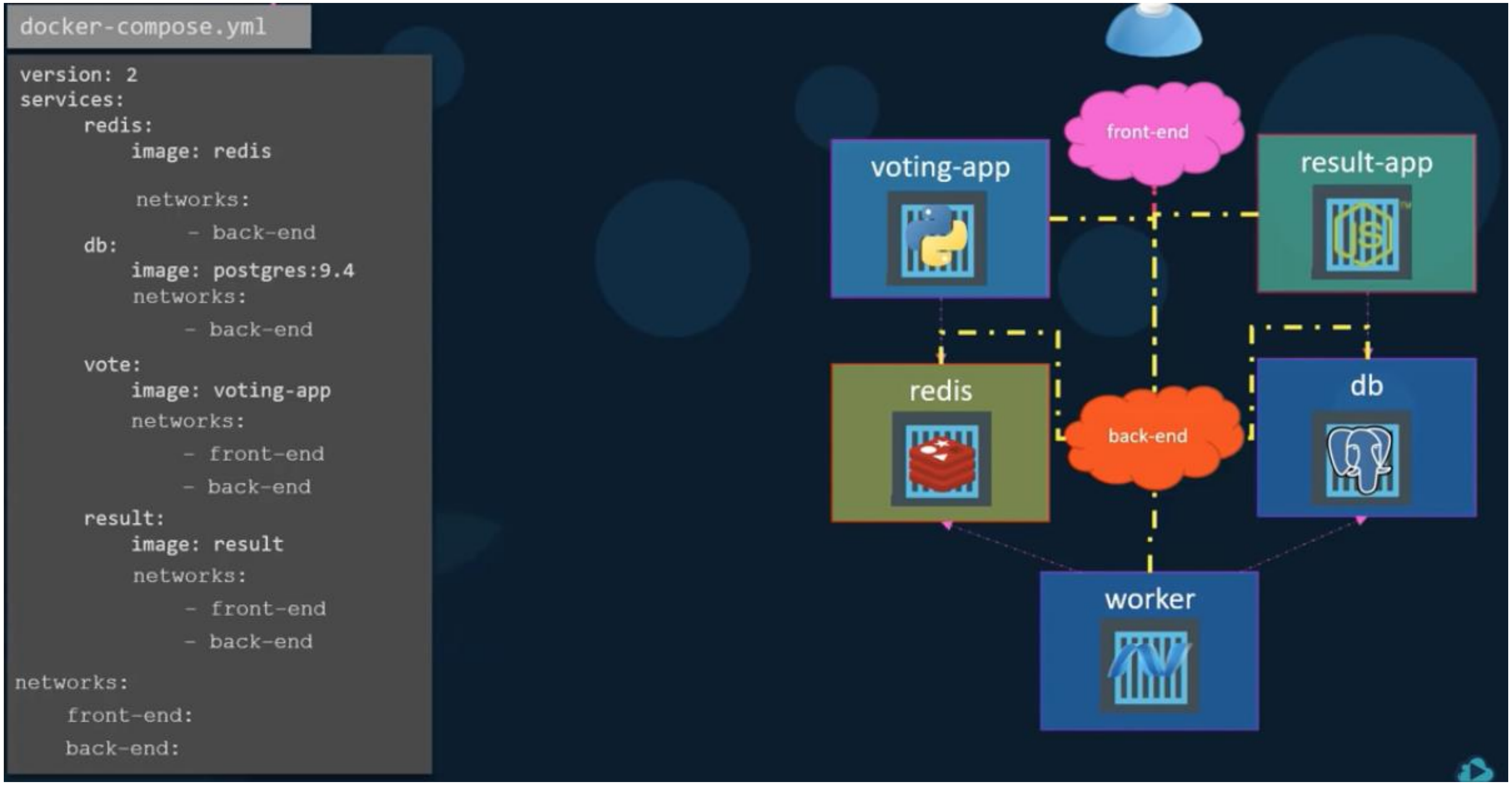
docker-compose.yml

```
redis:
    image: redis
db:
    image: postgres:9.4
vote:
    image: voting-app
    ports:
        - 5000:80
    links:
        - redis
```

docker-compose.yml

```
version: 2
services:
    redis:
        image: redis
    db:
        image: postgres:9.4
    vote:
        image: voting-app
        ports:
            - 5000:80
        depends_on:
            - redis
```

Docker Compose



docker-compose.yml

```
version: 2
services:
     redis:
         image: redis

          networks:
               - back-end
     db:
         image: postgres:9.4
         networks:
              - back-end
     vote:
         image: voting-app
         networks:
              - front-end
              - back-end
     result:
         image: result
         networks:
              - front-end
              - back-end
networks:
    front-end:
    back-end:
```