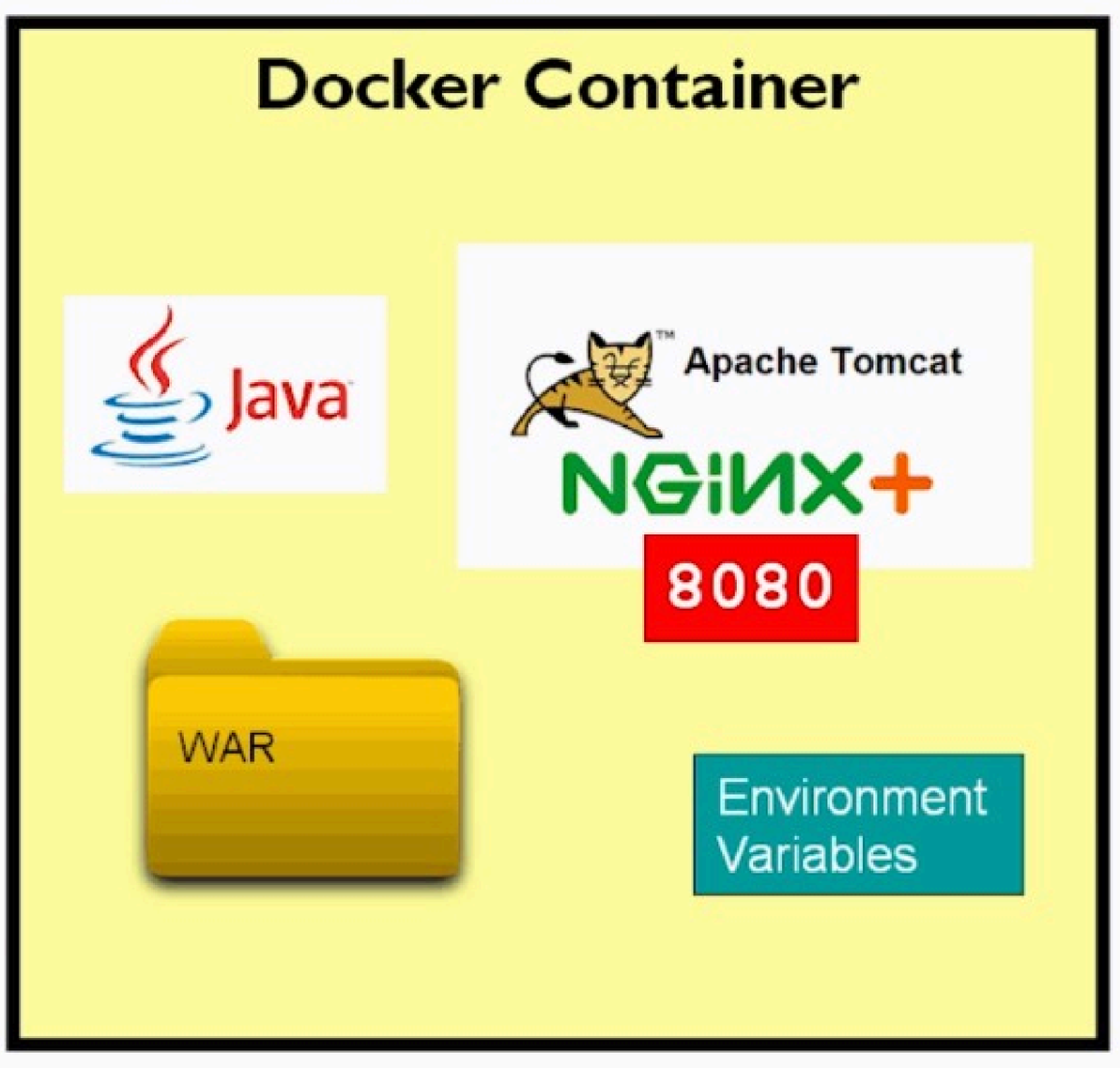
Docker

Docker for java developers



Agenda

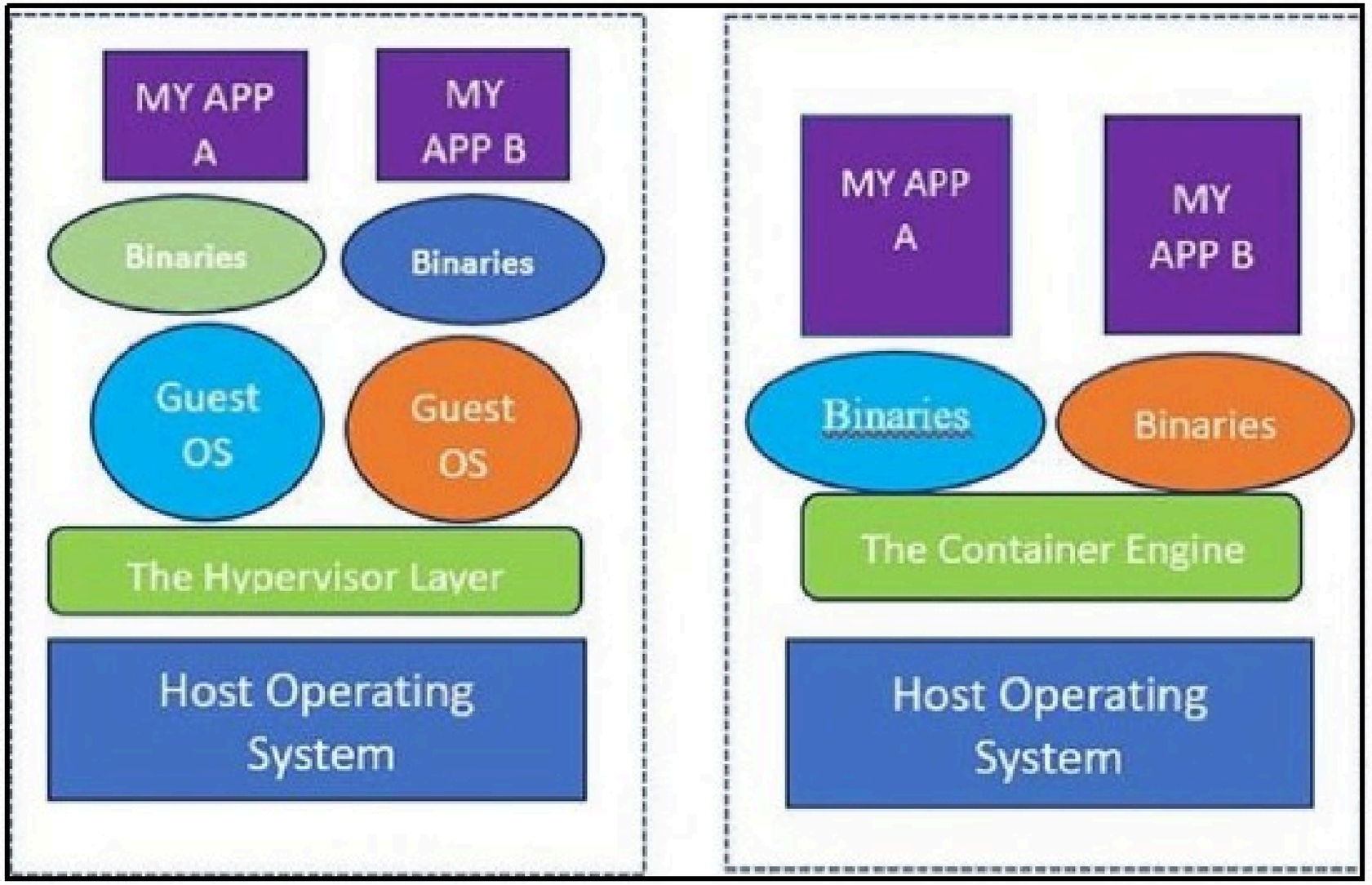
- Basic Introduction to docker
- Need of docker
- docker architecture
- Docker essential management commands
- Docker spring boot
- Docker networking
- Docker compose
- Kubernetes Tutorial for Beginners: Basics, Features, Architecture

What is Docker?

Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server.

This guarantees that the software will always run the same, regardless of its environment.

Docker vs VM


MY APP A
MY APP B
Binaries
Binaries
Guest OS
Guest OS
The Hypervisor Layer
Host Operating System
MY APP A
MY APP B
Binaries
Binaries
The Container Engine
Host Operating System

Concept of Containerization

The foundation of Containerization lies in the Linux Container (LXC) format. LXC is an operaeting system level virtualization method for running multiple isolated linux system (Containers) on an control host using a single Linux Kernel

The fundamental difference between VMS and Containers is that a container does not require its own OS, Thus making it more lightweight then a VM.

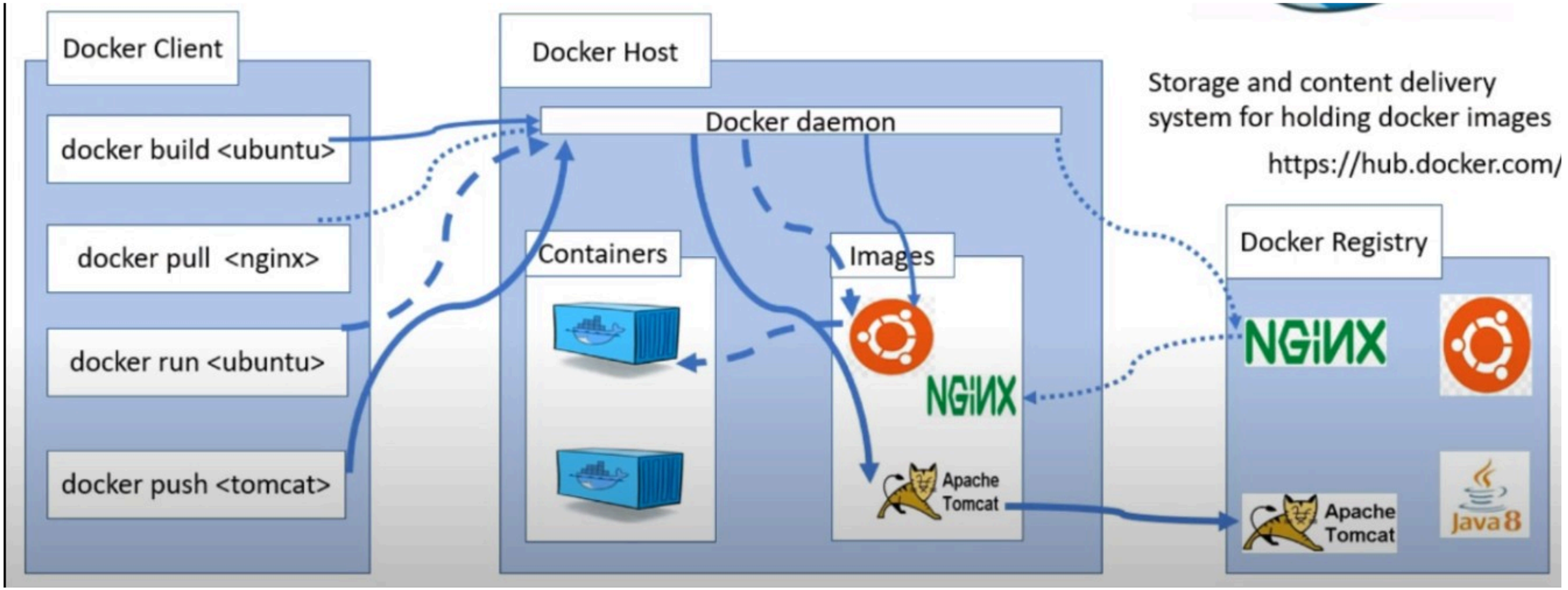
VM is slow to boot and take a long time to start up as compraed to Containers

Container is light weight and cost effective,as it share the machine OS kernel and dont required cost of associating an operating system with each application

Advantage of Containerization

- It works on my system:May be possible an lib is missing on production env
- Cost effective
- Light weight, performance
- Security, due to namespace, our app is going to sandboxed and dont communicate with
- each other unless we configure
- Dev can use exact same sw environment for both development and production
- Used with CI/CD pipeline integration, helping dev become more productive and efficient
- Support microservice arch
- Support scaling: scale up and scale down

Docker architecture



Docker Components

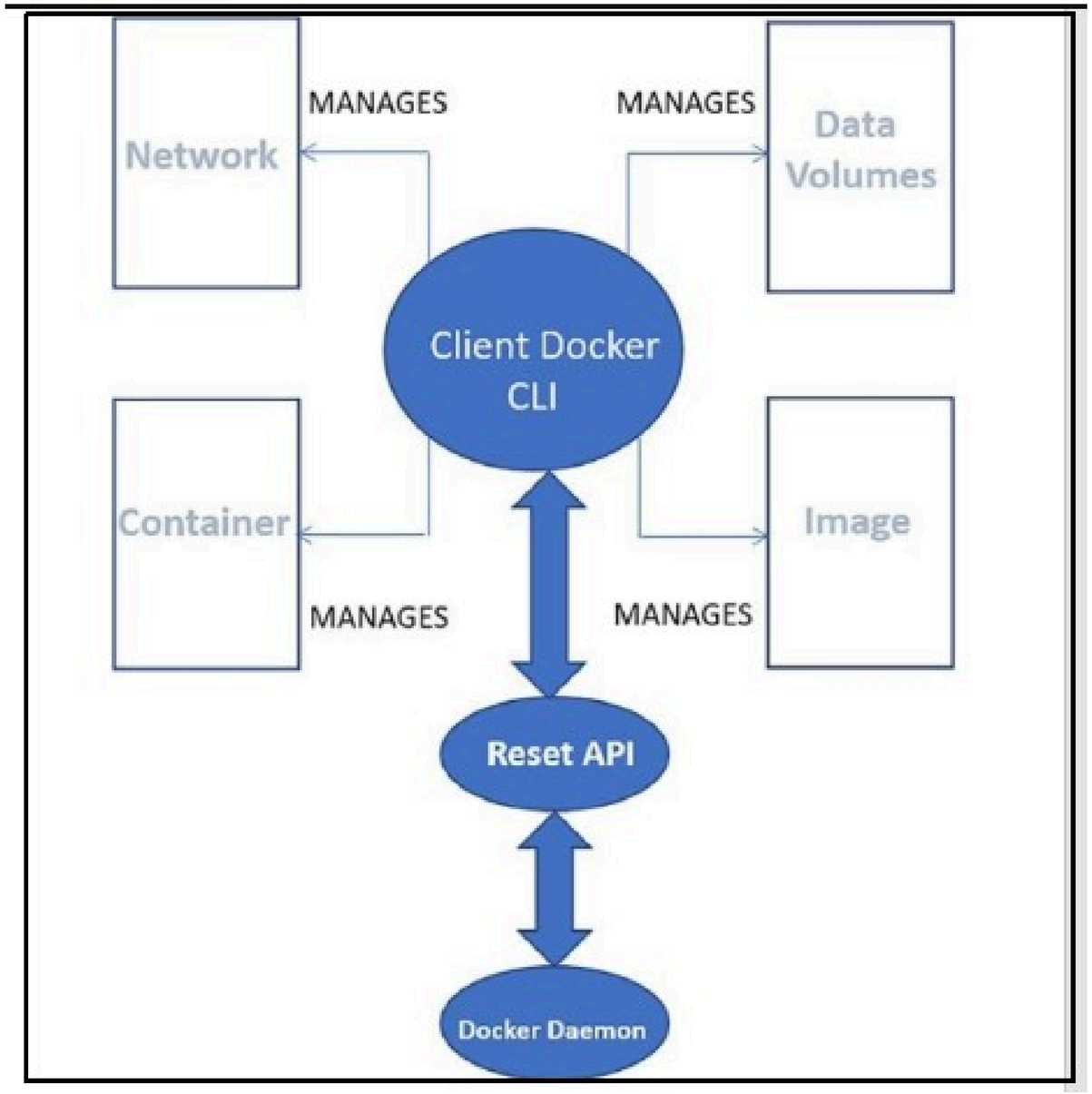
Docker is a piece of software that create, manages, administrates and orchestrates containers.
Docker is not a single monolithic application but made up of the components like containerd, runc and so on

The Docker Engine

The Docker Engine works on the principle of a client-server application. It can either be downloaded from *Docker Hub,* or we can build it manually from the source in GitHub. The Docker Engine can be deconstructed to have three main components:

1. The server, which is really the daemon process running.
2. The REST API, which takes stock of the programs that interface with the daemon process and instructs it about what is to be done.
3. And finally, the plain vanilla command-line interface-the the docker command.

Docker Engine component flow



Docker REST api provides a CLI for intracting with docker deamon

Docker daemon is responsible for heavy lifting of creating, administrering and managing docker containres, images, network storages, volume etc

Docker Hub and Docker Registry

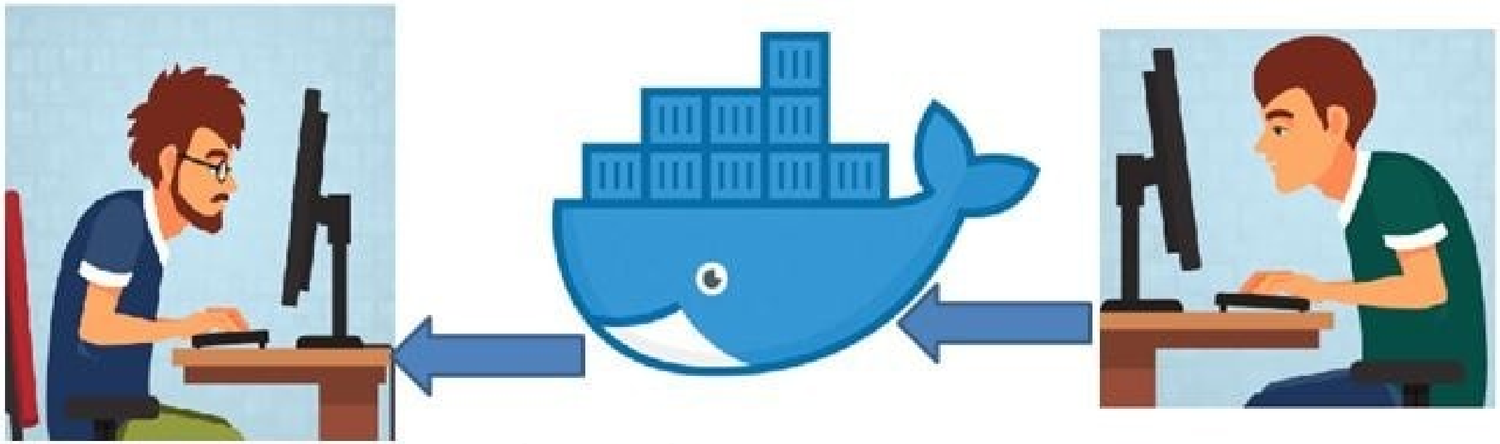
A Registry defined as storage and contect delivery system.
They hold docker images which are available in different tagged versions.
Docker Registry is a service that is holding our docker images.
Docker registry could also be hosted by a third party, as a public or private registry,
Ex: Docker hub, AWS Container Registry, Google container registry etc

WhyDocker?

Developers use Docker to eliminate “works on my machine” problems when collaborating on code with co-workers.

Operators use Docker to run and manage apps side-by-side in isolated containers to get better compute density.

Enterprises use Docker to build agile software delivery pipelines to ship new features faster, more securely and with confidence for both Linux and Windows Server apps.



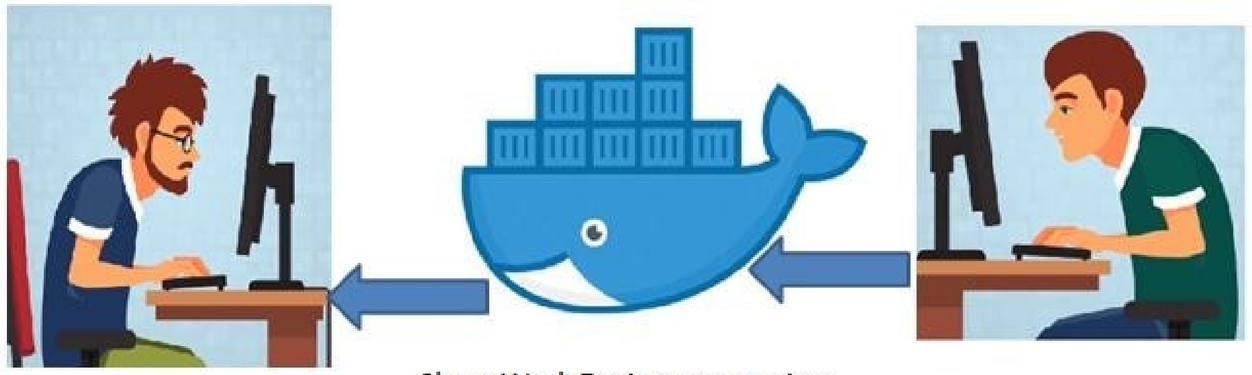
Share Work Environment using Docker

What is Docker?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.

Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

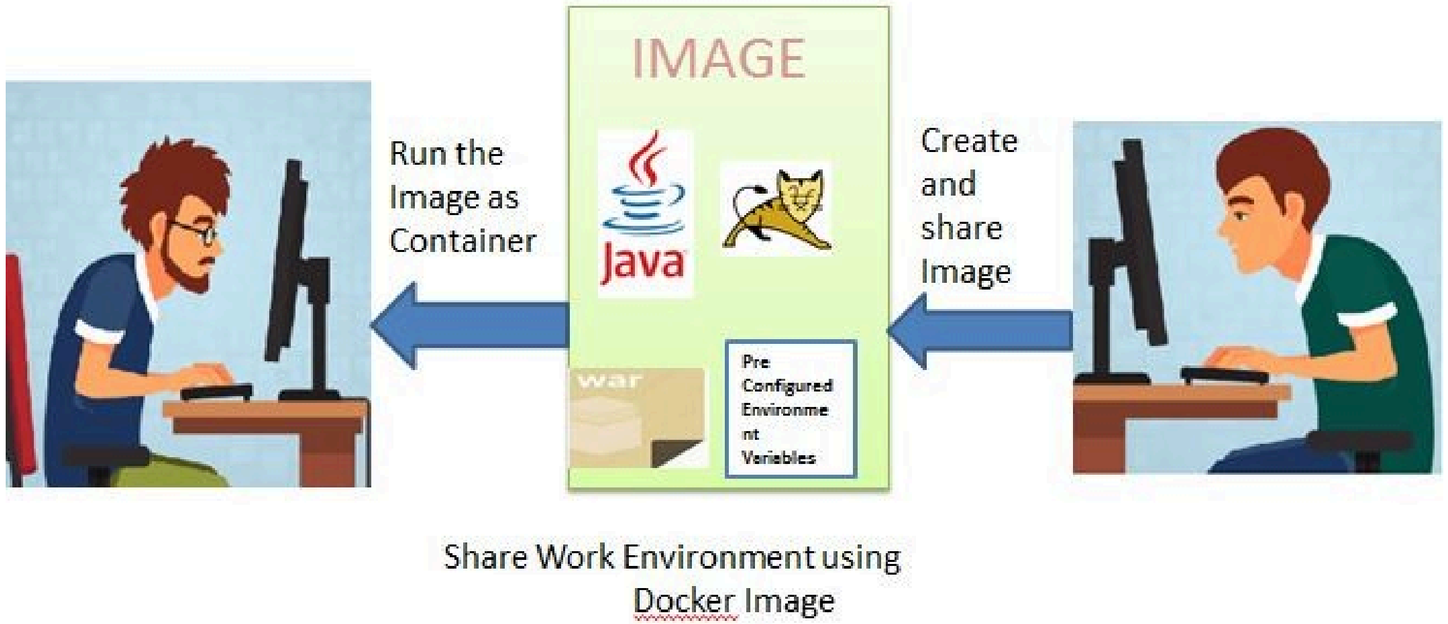
Create self-contained development environments inside Docker containers.
So we share an environment already configured.



Share Work Environment using Docker

Docker Image

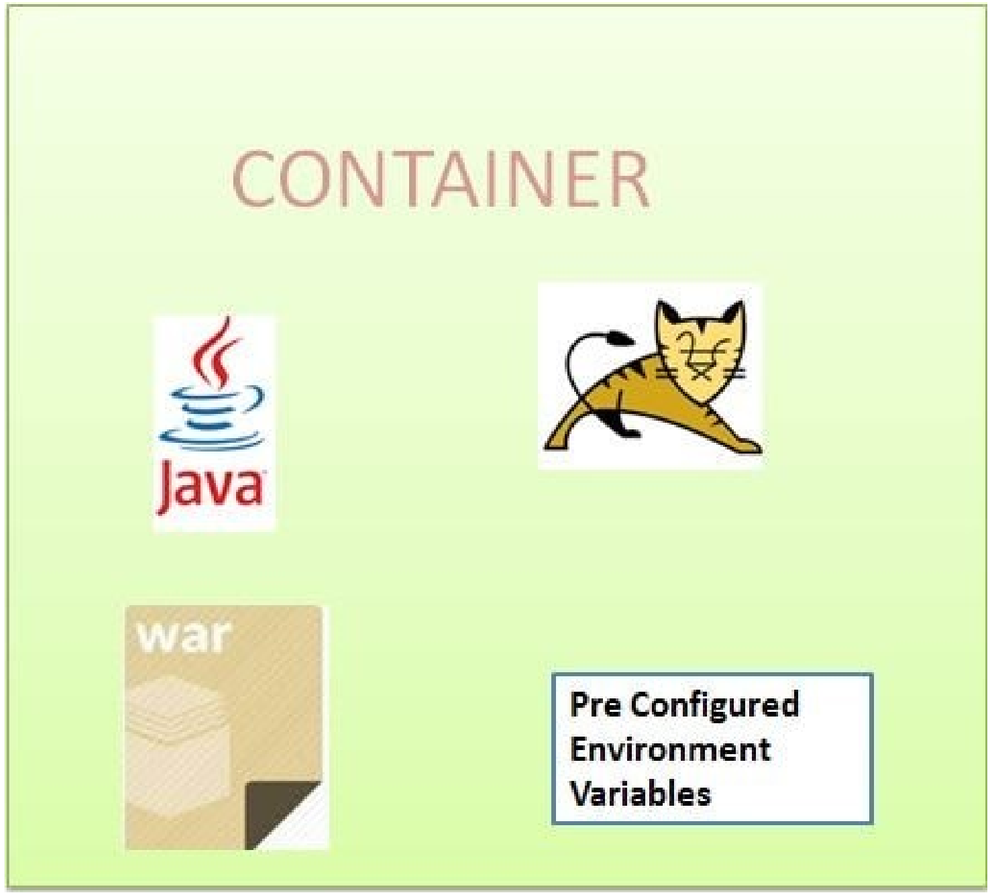
An image is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files
So an existing fellow developer will create an image of his environment and share it with the new developer. The new developer will just have to run the image as a docker container.



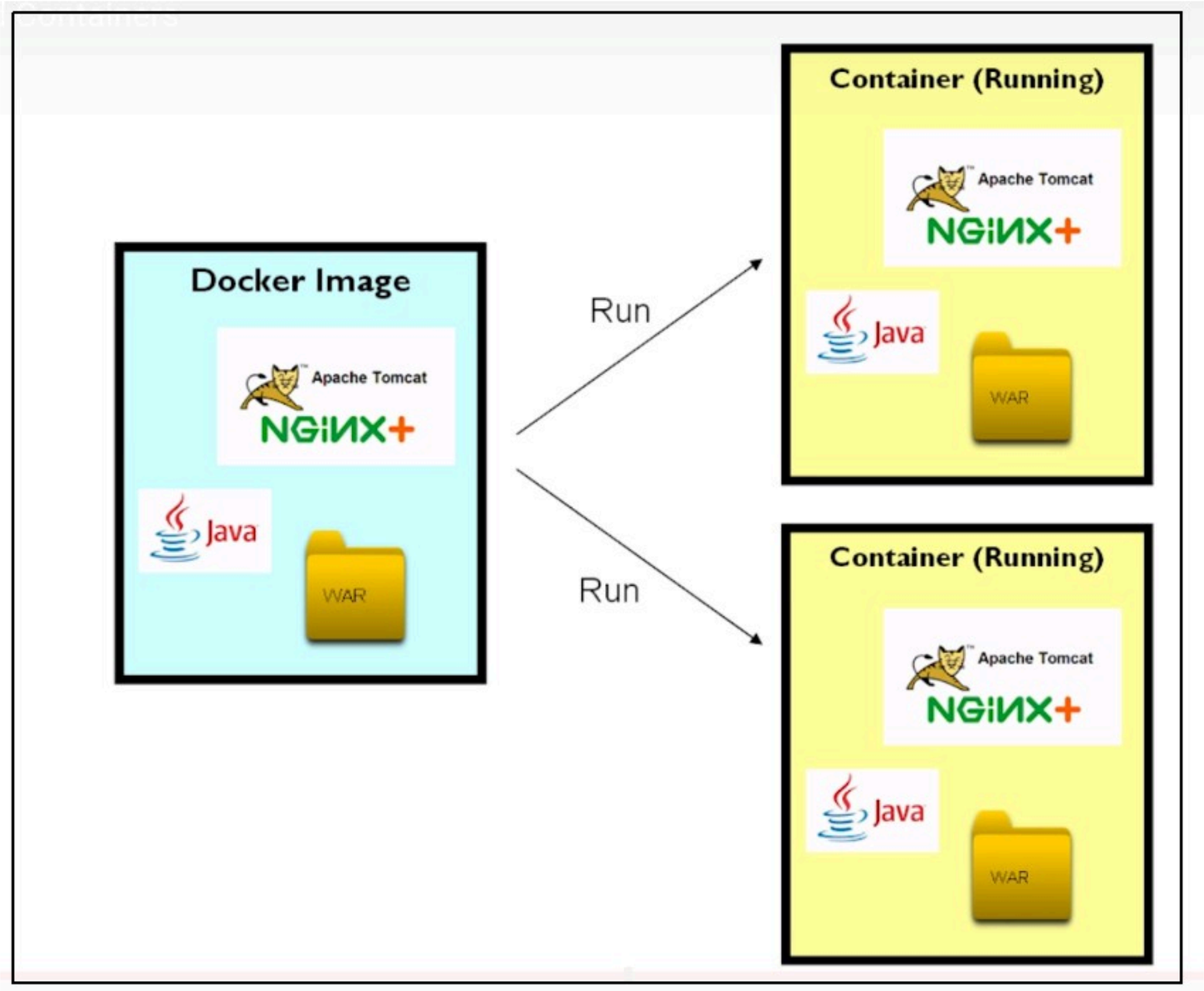
Share Work Environment using Docker Image

Docker Container

A container is a runtime instance of an image what the image becomes in memory when actually executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.



Docker image and containers



use of docker for Java Developers

Sharing development workspace, with preconfigured development environment.
Continuous integration is one of the most popular use cases for Docker.
Teams looking build and deploy their applications quickly use Docker, combined with ecosystem tools like Jenkins, to drive apps from dev, testing staging and into production without having to change any code.

Docker essential Commands

docker -v: checking docker version

docker --version

docker info: give complete info about docker, how many container running, pause etc

docker –help: give help about each command

docker images: give details of all images on local docker installation

docker images --help

Docker images/containers Commands

docker images: give details of all images on local docker installation

docker images –help

docker pull ubuntu:latest

docker image rm <id>

docker container ls -a

docker container run -it ubuntu ctr+q ctr+q

docker container stop <id>

docker container start <id>

docker ps -a

docker rm <id> removing all containers

docker rmi -f $(docker images -a -q) remove all images

docker system prune delete everythings

Docker pulling and running images

Ubuntu image

docker pull ubuntu:latest

docker container run -it ubuntu

apt-get update && apt-get install git

apt-get install openjdk-11-jdk

Tomcat image

docker image pull tomcat:latest

docker contrainer run -d -p 8080:8080 tomcat

(stop any container running on port 8080)

docker container log -f <id>

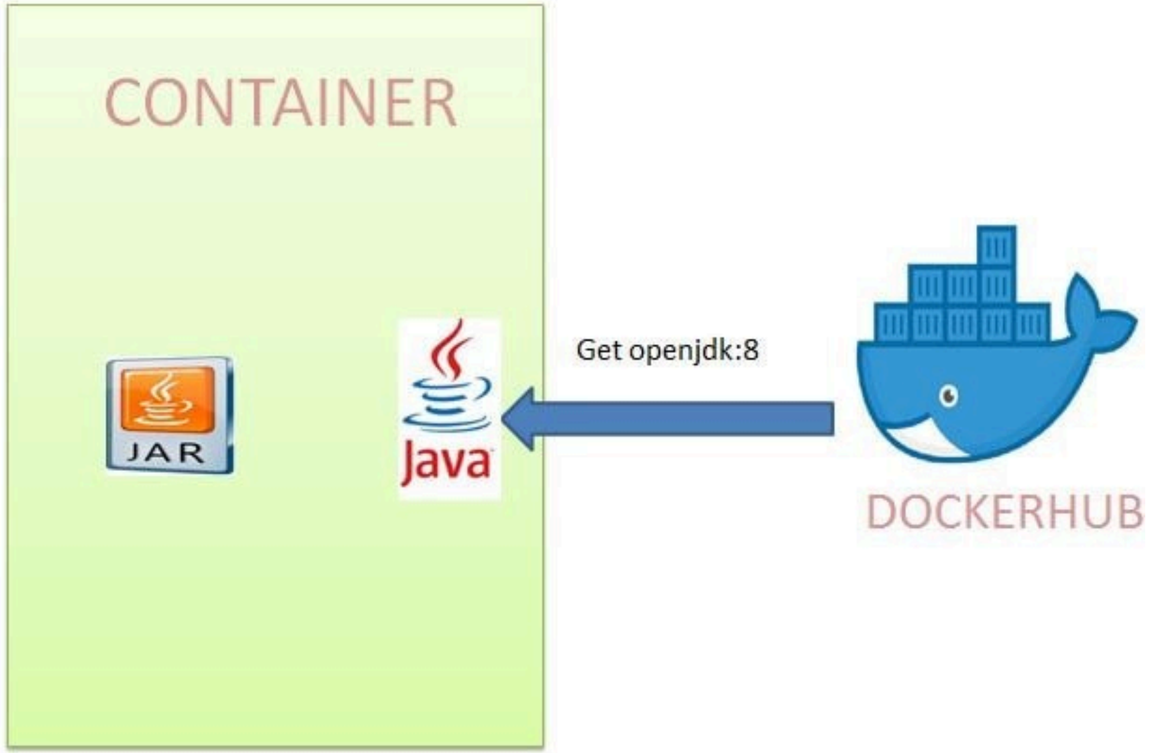
Docker file

Docker file is a normal text file used to build our own docker images by giving specific instructions, so that we can create our own customized docker images in an automatic way without running a docker container

```
FROM ubuntu
MAINTAINER rajeev gupta <rgupta.mtech@gmail.com>
RUN apt-get update
CMD ["echo","welcome to docker file"]
```

```
raj@raj-Aspire-E5-575G:~/Desktop/demo$ docker build -t my_docker_file .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM ubuntu
 ---> 4dd97cefde62
Step 2/4 : MAINTAINER rajeev gupta <rgupta.mtech@gmail.com>
 ---> Running in 59daef82f47c
Removing intermediate container 59daef82f47c
 ---> ce9457c9f366
Step 3/4 : RUN apt-get update
 ---> Running in 17de3eaa4651
```

Spring boot docker hello world



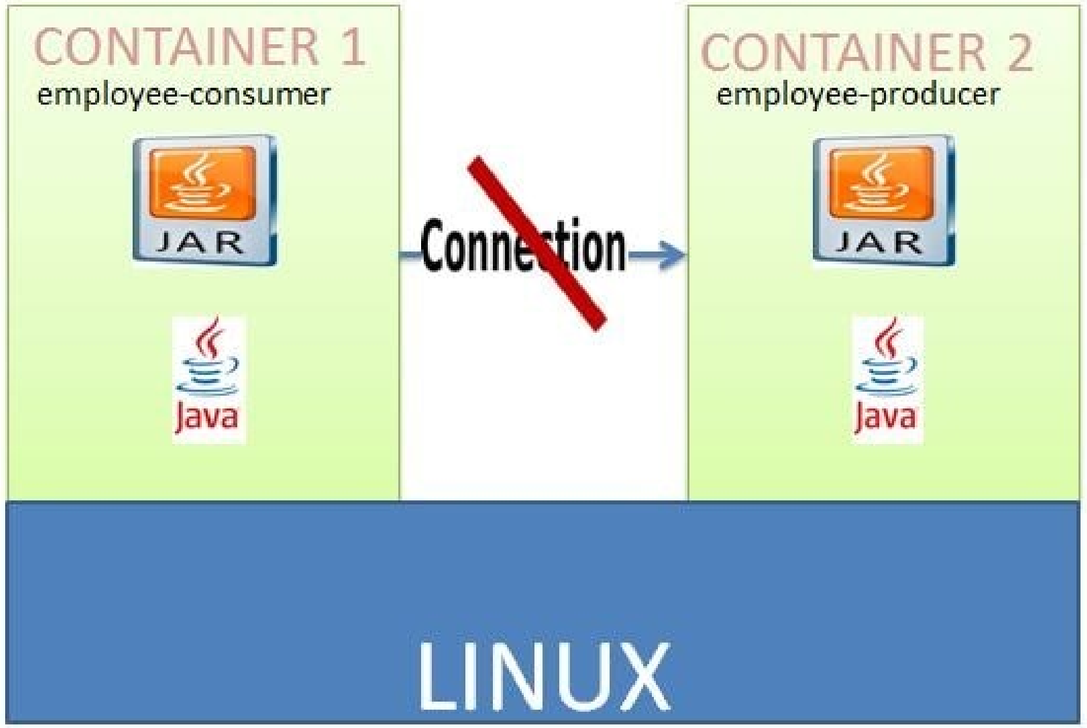
Docker networking

The way Docker has been designed such that a Docker Container should have only a single service running.

Again we can have multiple services running in docker using some workarounds but this will not be a good design.

So we will be deploying the two microservices employee-producer and employee-consumer to
two different containers and then have them interact with each other.

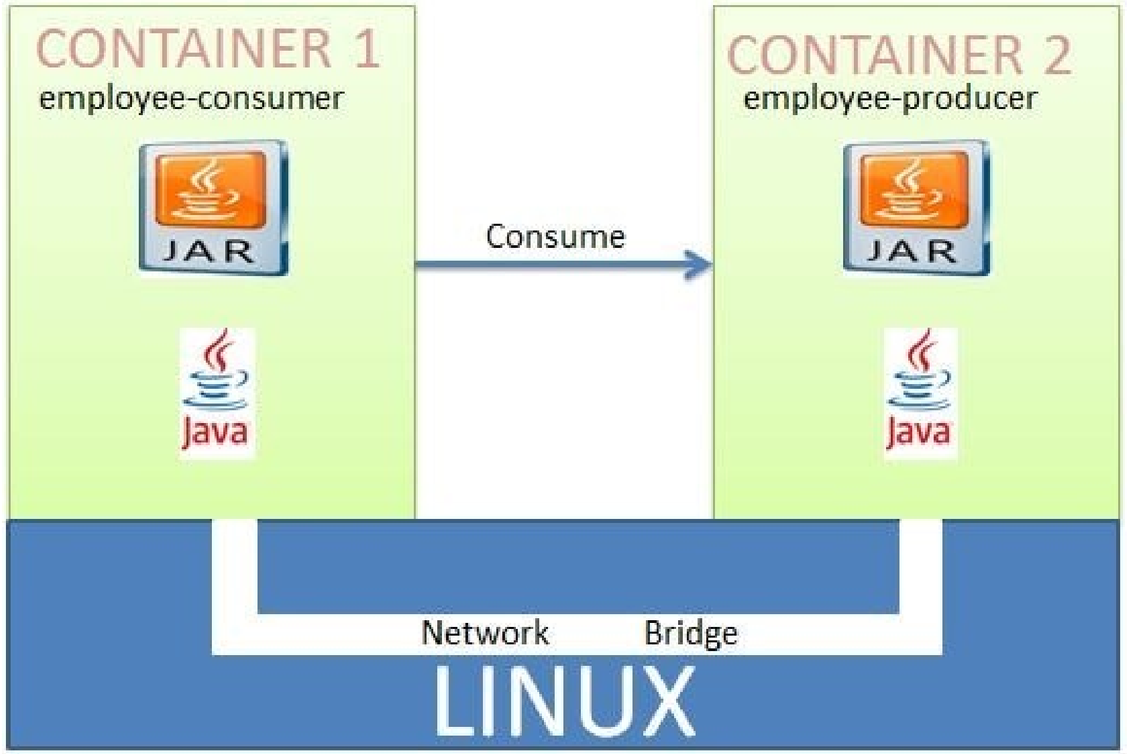
In order to achieve this will have to make use of the docker networking commands.



Inter Docker Container Communication Using Docker Networking

We will be using Docker Networking to allow multiple containers to interact with each other.

We will need to create our own network and add both the employee-producer and employee-consumer services to it.

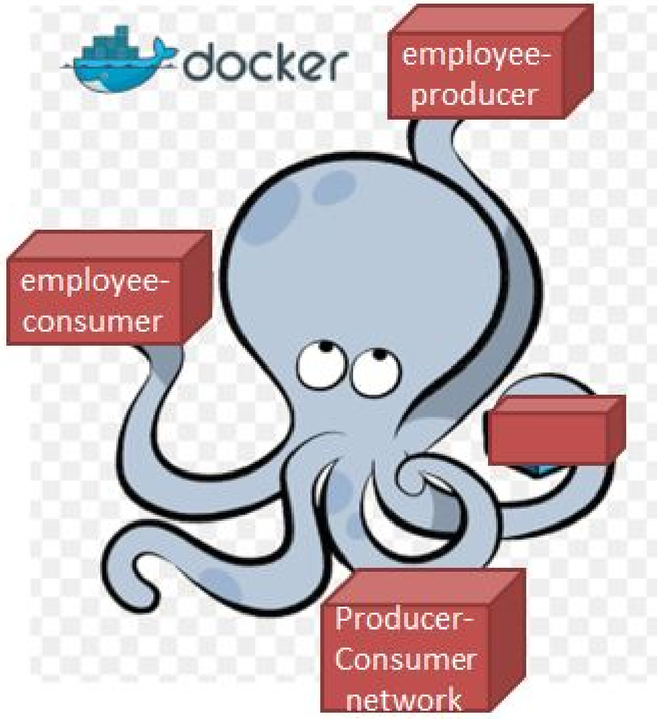


Deploying Multiple Spring Boot Microservices using Docker Compose

Compose is a tool for defining and running multi-container Docker applications.

With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Using docker-compose we will be creating the custom network named consumer-producer and then starting the containers employee-producer and employee consumer.



Using docker-compose we will be creating the custom network named consumer-producer and then starting the containers employee-producer and employee consumer.