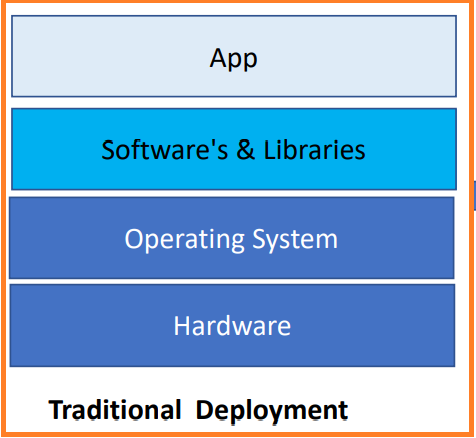
**Introduction:**

* **Docker** is a containerization platform which packages your application and its all dependencies together in the form of containers. So as to ensure that your application works seamlessly in the environment be it dev or test or production.
* **Docker is a software which use the less resource and run application in any were.**
* Using this we can build ship and run applications as containers
* **Docker available in two editions**: ***Community Edition (CE) and Enterprise Edition (EE).***
* Docker is cross platform.
* **Container: Docker container** Contains application code + its dependencies (software configuration +ENV) etc.
* If we build application using docker, it will easy for us to move application from one location to another location.

**Deployment Approaches (Before Docker):**

* In Traditional Deployment model we need Hardware, **CPU,RAM,NETWORK**
* On TOP of ***Hardware*** we used to install ***Operation system(i.e interface b/w the hardware and*** *software)*
* We ***used to install linux,windows operating system on top of hardware machine***
* To deploy any application we used to configure software’s/Libraries
* We can HOST application on top of configuration

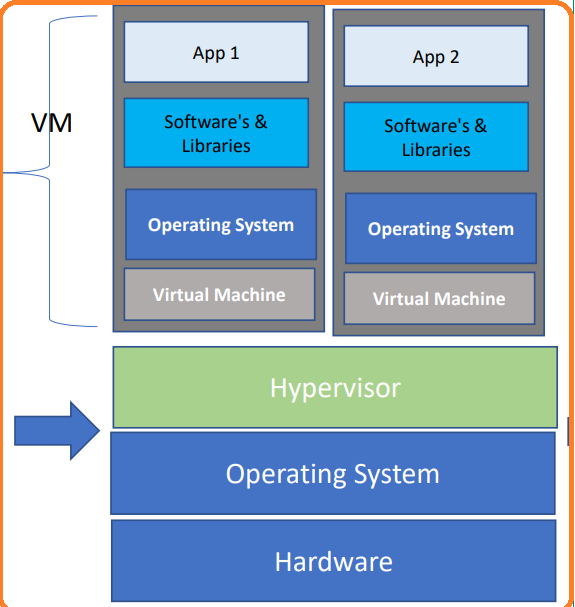
****

**Problems in Traditional Deployment:**

* Installation and configuration
* Time consuming and not utilizing infrastructure properly (we can deploy multiple application in same kind of environment because process isolation)
* Need to perform install and configuration across all the servers (i.e QA, DEV, PREPROD..ETC)
* Without installation and configuration cont deploy application
* Compatibility issues, If we miss any configuration (i.e if we install java9 instead of java8)in this case application will be work on one environment and other environment will not work
* We are not utilizing hardware resource as much as possible

**Virtualized Deployment:**

* Due to above issues, people started using ***“virtualized deployment”***
* We can create virtual machines based on physical machine capacity
* Can create multiple virtual machines using *“****Hypervisor” software on top of the physical machine***
* Each virtual machines will have its own operating system(***i.e GUEST Operating system***)
* Using ***HYPERVIZOR*** we are doing hardware level isolation
* On top of this we can deploy application
* We are able to utilize the hardware resource more efficiently
* Each application will have own dependencies
* **Network level isolation:** Can run application one VM with port number 8080 other application will run with same port number 8080 without any port conflict is called Network level isolation same thing we can achieve in Traditional approach will get port conflict issues.



**Hypervisor:**

* Allow an operating system to run independently from the underlying hardware through the use of virtual machines.
* Share virtual computing, storage and memory resources.
* Can run multiple operating systems on top of one server (bare-metal hypervisor) or installed on top of one standard operating system and isolated from it (hosted hypervisor).
* Using this Hypervisor we are isolating the hardware**[cpu,network,memory,storage etc]**

We have two types of Hypervisor

**Type-1 Hypervisor:**

* Will run directly on top of ***Hardware***
* We can called as ***bare metal server***
* We don’t need ***HOST*** operating system
* ***Real time will use TYPE-1 Hypervisor***

**Example:**

* VM Ware
* ESI Hardware
* Dell Servers
* Microsoft hypervisor/ZEN

**Type-2 Hypervisor:**

* Will required operating system to
* On top of operating system will run application
* AWS is using “***CITRIX ZEN”*** Hypervisor software

**Example:**

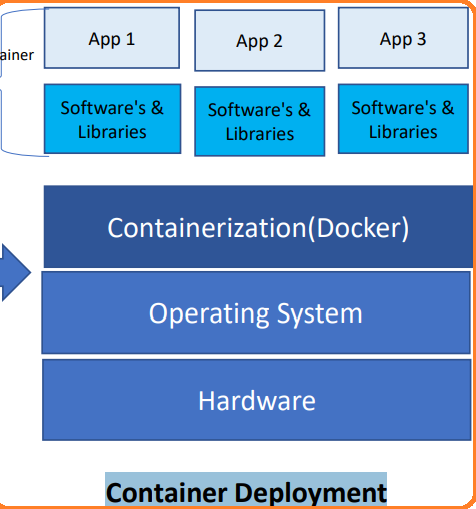
* + - Oracle Virtual Box
    - VM Ware work station
    - VM Ware player

**Problems in Virtulization Deployment:**

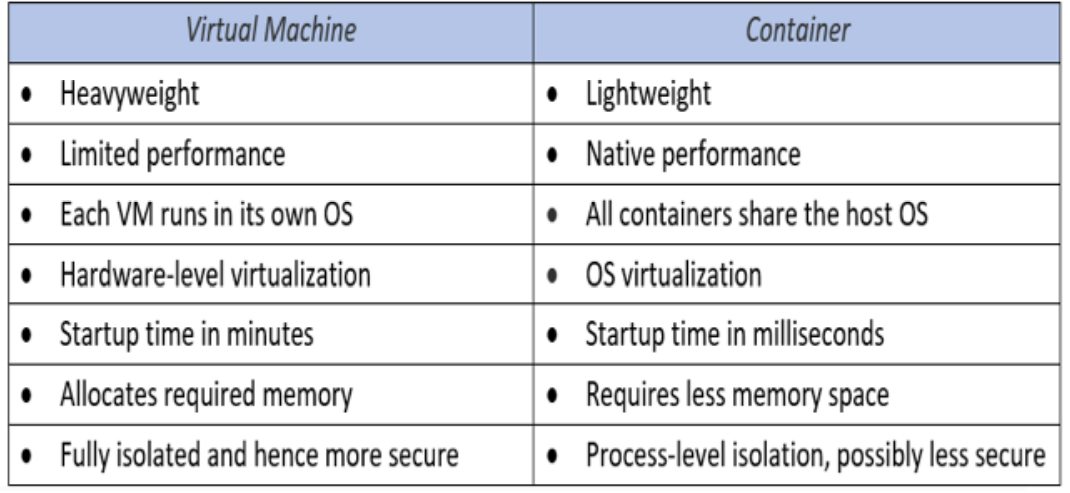
* Required ***GUEST*** operating system every ***VM***
* Virtual Machines are very slow
* Needs a **hypervisor** application to create virtual machines
* VM’s are isolated at user level, Network level and etc
* We can’t host other application in same physical machine even though we have enough RAM and storage because application need isolated.
* Virtual machines are **heavy weight** because we will install operating system in each and every system
* It will take lot of resources like CPU, memory and storage from host machines
* VM’s are slow
* Within VM we can’t isolate another machine
* **Compatibility/version** issues (i.e application is working on environment and other environment will not work)

**Container Deployment**

* To overcome above issues, people started using Containerized deployment
* The way we have Hypervisor in virtual deployment mode same thing we have containerization software
* Popular containerization software in market ***(Docker, Podman,ContainerD,CRIO,Racket(rkt)***
* Using containerization software we will create a containers(the popular containerization software in Docker)
* Container will carry application code and dependencies
* One application will isolated to another application with container(i.e Operating System level isolation)
* Container does not have complete operating system and ***common for all container is HOST machine operating system called as kernel***
* We can deploy multiple application in single virtual machine using containerization software
* Containers are light weight
* No operating system in container that’s why it’s light weight
* Does not required guest operating system as its shares host OS between containers
* Does not required ***HYPERVISOR***
* Docker container packages are self contained and can run application in any environment



* **Kernel** is mediator between *containerization and hardware*
* **Kernel** is common for all containers
* **Kernel** will convert the code human readable to machine readable code
* Each container will have its own process storage and host operating system common for all the container
* Containers are light weight component
* Containers are portable
* We can shift the container anywhere



We have other ***containerization platform*** which widely used in market.

**Example:**

* **Docker**
* Container –D
* CRI-O
* Rocket(RKT)
* PodMan

**Docker has 2 Edition:**

* **Docker Community Edition (DCE):** Docker CE will not be supported by Redhat(i.e most of companies is using community edition only)
* **Docker Enterprise Edition (DEE):** Docker EE will be support most of the OS including redhat.
* This 2 features will support in DEE compare with DockerCommunityEdition(DCE)
* **DTR(Docker Trusted Registry):** Private Repository to store / maintained docker images
* **UCP(Universal Control pane)**: GUI for manage containers & Docker nodes

**Why Containers?**

**Developers care because:**

* Quickly create ready-to-run packaged applications, low cost deployment and replay
* Automate testing, Integration, packaging(***We can automate the process of creating docker packages using jenkins***)
* Reduce eliminate platform compatibility *issues (i.e application is working one environment not working other environment)*
* If application is working one environment it should work in other environment
* Integrate container using CI/CD
* Containers will support micro services
* Containers are best ways to run micro services

**IT cares because:**

* Improve speed and frequency of releases, reliability of deployments
* Makes app lifecycle efficient, consistent and repeatable -configure once, run many times
* Eliminate environment inconsistencies between development, test, and production
* Improve production application resiliency and scale out /in on demand
* In real time 99% will be Linux environment

**Note:** *Recently* ***MIRANTIS*** *acquire docker.*

|  |  |
| --- | --- |
| **Type** | Containerization Tool |
| **Vendor** | Docker |
| **Is Open Source** | Yes- Some extent |
| **Operating System** | Cross Platform |
| **URL** | www.docker.com |

**Docker Architecture**

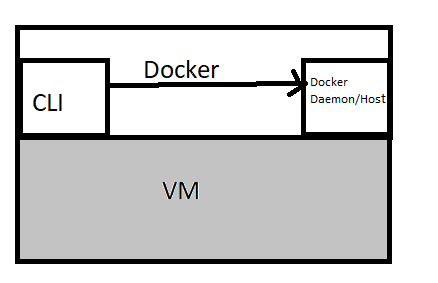
* We have 3 major components in docker

**Docker Client:**

* Using CLI we can interact with docker
* We can execute some docker commands using CLI
* CLI will send the instructions /communicates to the docker daemon
* Docker daemon will process the request

**Commands can be:**

* Creating images/containers
* Push images/pull images from registry
* Push image to registry



**Docker Daemon(docker process):**

* Will execute the commands send by the user
* Command can be creating image and push image etc

**Docker Registry(i.e To store some files and packages):**

* ***It’s a central repository where we can maintain docker packages(docker Image)***
* It’s a collection of repositories. In Docker registry we can create multiple repositories
* Central repository maintained by Docker and Docker hub. This registry contains various public images that can be used. Also user can create his own image and upload the image in registry either as public or private.
* We can create a repositories and maintain the docker images in below artifacts tools
* Collection of **repositories called Registry**
* Nexus
* JFrog
* ECR(AWS) Elastic container Registry

**Public Registry:**

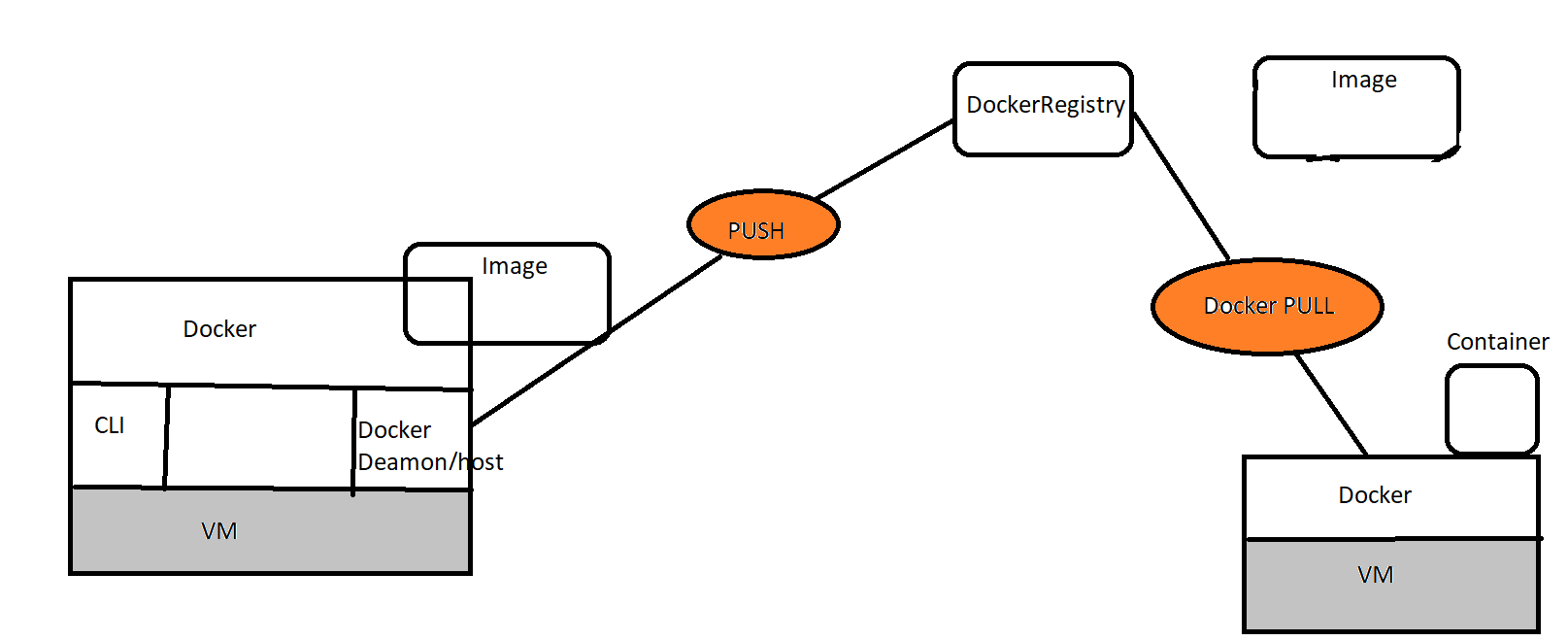
* **Docker hub is a public repository:** Which contains all the open source software’s as a docker images. We can think of docker hub as play store for docker image

**Private Registry:**

* **NEXUS**
* **JFrog**
* **ECR:** Elastic container registry
* **ACR:** Azure Container Registry
* **GCR:** Google Container Registry
* **DTR:** Docker Trusted Registry
* We can store and share the docker images with in our company network using private repo

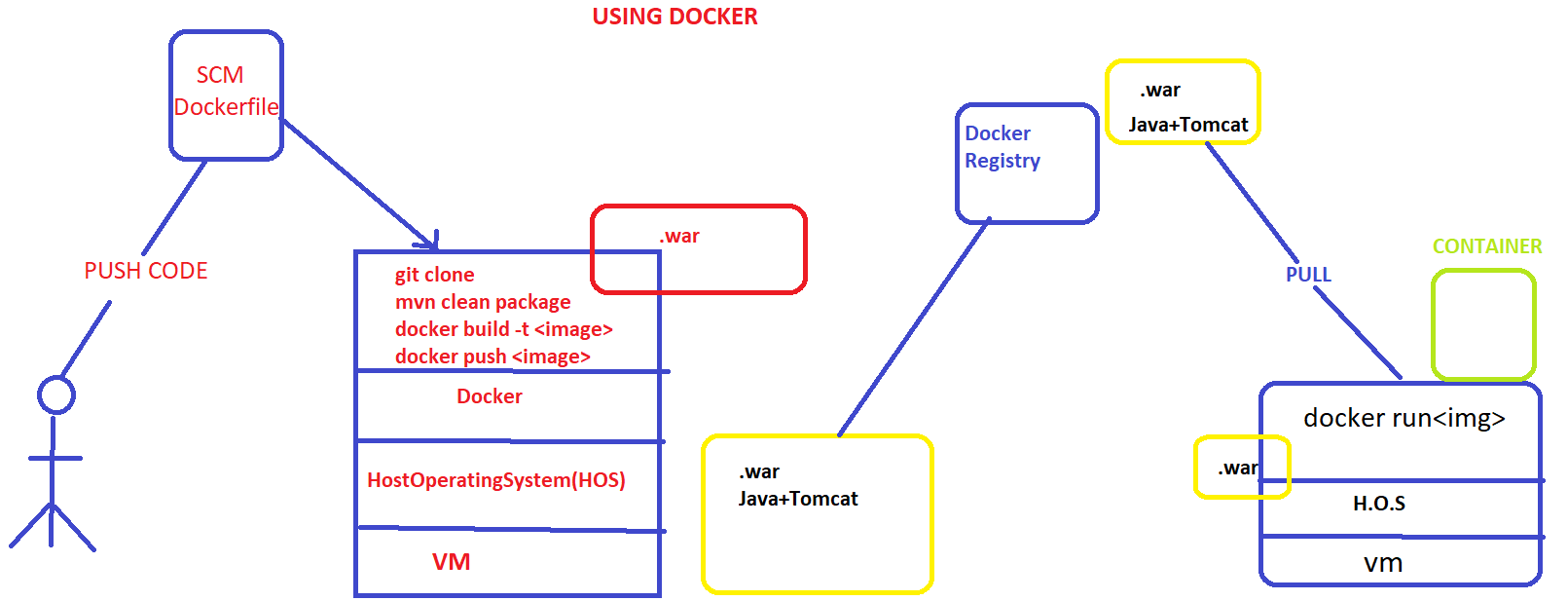
**Docker Repositories:**

* ***Docker repository is collection of images with same type (application) and multiple tags (Versions).***



**Deploy application using Docker:**

* Developer will commit the code
* Build the package suing MAVEN
* Create one more package using Docker
* Docker image contain Application code software (required)
* To run .war file we need java/jboss/Tomcat
* We need to write dockerfile
* Using Dockerfile will create a image
* From base image we will get java/Tomcat
* Push image to registry using docker command
* Create container based on image
* If its works in one environment will work on other environment
* If we have other application we can create container without impact of other container



**Containerization**

* Docker is containerization tool
* In comparison to the traditional virtualization functionalities of hypervisors,
* Docker containers eliminate the need for a separate guest operating system for every new virtual machine.
* Docker implements a high-level API to provide lightweight containers that run processes in isolation.
* A Docker container enables rapid deployment with minimum run-time requirements. It also ensures better management and simplified portability.
* This helps developers and operations team in rapid deployment of an application.
* No Fixed hardware

**Virtualization:**

* Fixed hardware allocation
* required Guest operating system

**Key Terminology:**

1. **Docker image:**

* It’s just package which contains everything (application code & Software’s, Libraries, Configurations,ENV) whatever is required to run piece of software.

1. **Docker Container:**

* It’s a runtime instance (process) of docker image
* If we run docker image container (process) will create
* Like without AMI we can’t create EC2 instance same without docker image we will not create container.

1. **Dockerfile:**

* Its text file which container some instruction to create image using this docker file will create image from the image will create container.
* Simple word, we can say dockerfile is build script to create docker image

1. **DockerImage:**

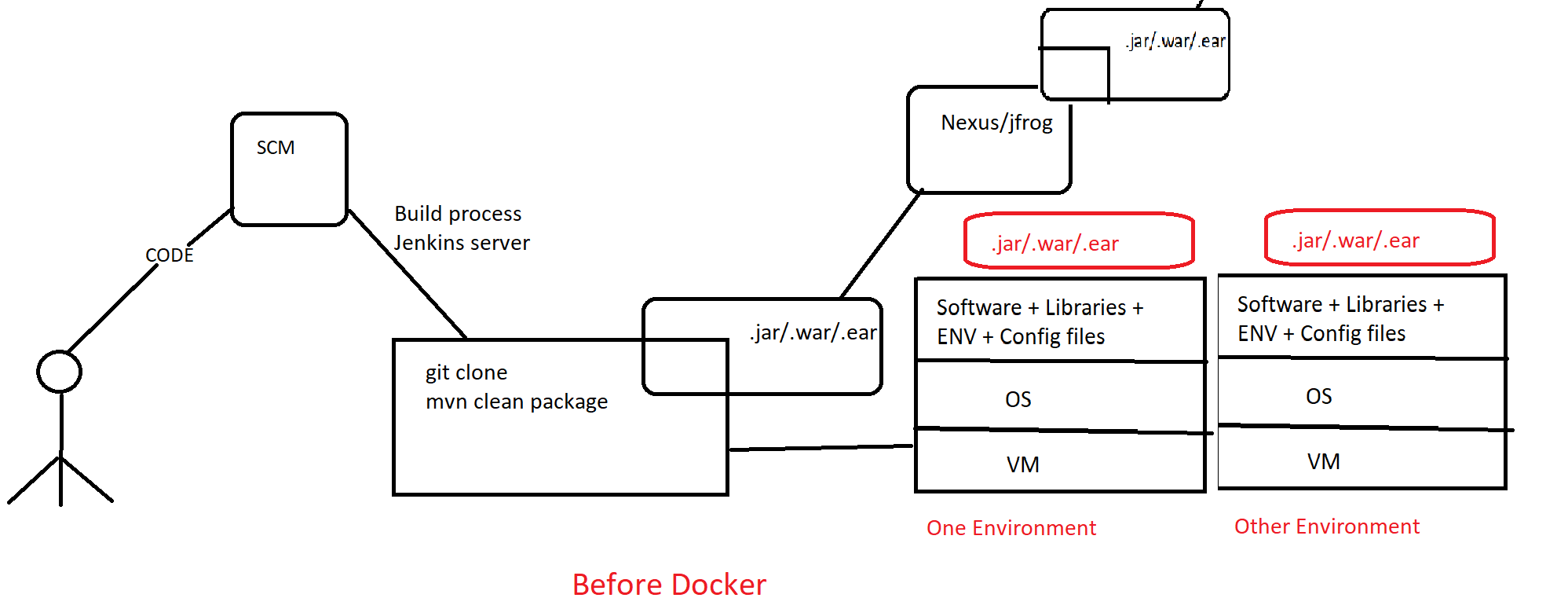
* Docker image is lightweight executable package that container everything piece of software

**DockerFlow:**

* Using Dockerfile we will build the image, image is not have complete operating system and it is having libraries interact with host operating system ***(i.e kernel)***
* Image and container will not have complete operating system
* ***From dockerfile will create a image 🡪 run the image will create container 🡪 run the image will create container***, if container is up and running then part of application will be up and running.

**Advantages:**

* **Rapid application deployment**
* **Portability across machines**
* **Version control and component reuse** : will maintain all the version of docker images/rollback if current image not working.
* **Sharing**
* **Lightweight:** not having complete operating system
* **Self-sufficient**



**Using Docker deployment:**

1. As per build process we will build the application using docker

2. We will create image using docker build

3. We will store image in docker registry (central registry)

4. Pull the image from registry and create the container

5. Once the deployment done, we can move the application from one server (i.e dev) to other server (i.e test server) without any issue and also deploy multiple container with same machine with isolated.

.