NPTEL ONLINE CERTIFICATION COURSES

Cloud Computing

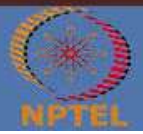
Prof. Soumya K Ghosh

Department of Computer Science and Engineering

Module 10: Cloud Computing Paradigm
Lecture 45: Cloud Migration - I

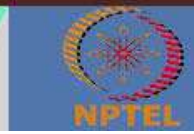
CONCEPTS COVERED

- VM Migration - Basics
- Migration strategies



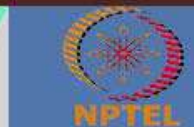
KEYWORDS

- Virtual Machine (VM)
- VM Migration



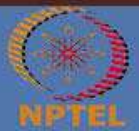
VM Migration

NPTEL

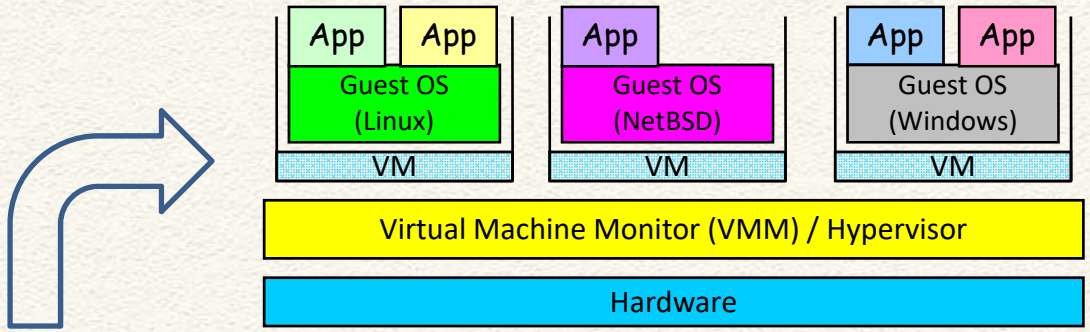


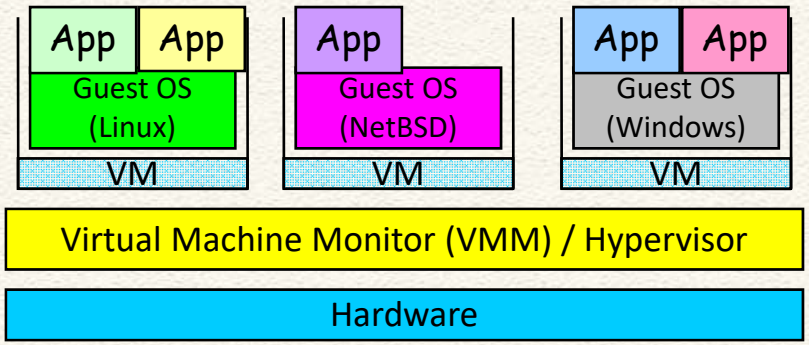
VM Migration

- VM Migration – It is process to move running applications or VMs from one physical server/ host to another host.
- Processor state, storage, memory and network connection are moved from one host to another host
- Why to migrate VMs?
  - Distribute VM load efficiently across servers in a cloud
  - System maintenance



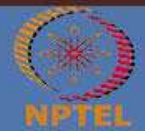
Virtualization





VM Migration – Needs

- **Load Balancing**: For fair distribution of workload among computing resources.
- **Maintenance**: For server maintenance VMs can be migrated transparently from one server to another.
- **Manage Operational Parameters**: To reduce operational parameters like power consumption, VMs can be consolidated on minimal number of servers. Under-utilized servers can be put on a low power mode to reduce power consumption.
- **Quality-of-Service violation**: When the service provider fails to meet the desired quality-of-services (QoS) a user can migrate his VM to another service provider.
- **Fault Tolerance**: In case of failure, VMs can be migrated from one data center to another where they can be executed



VM Migration – Types

- **Cold or Non-Live Migration:** In case of cold migration the VM executing on the source machine is turned off or suspended during the migration process.

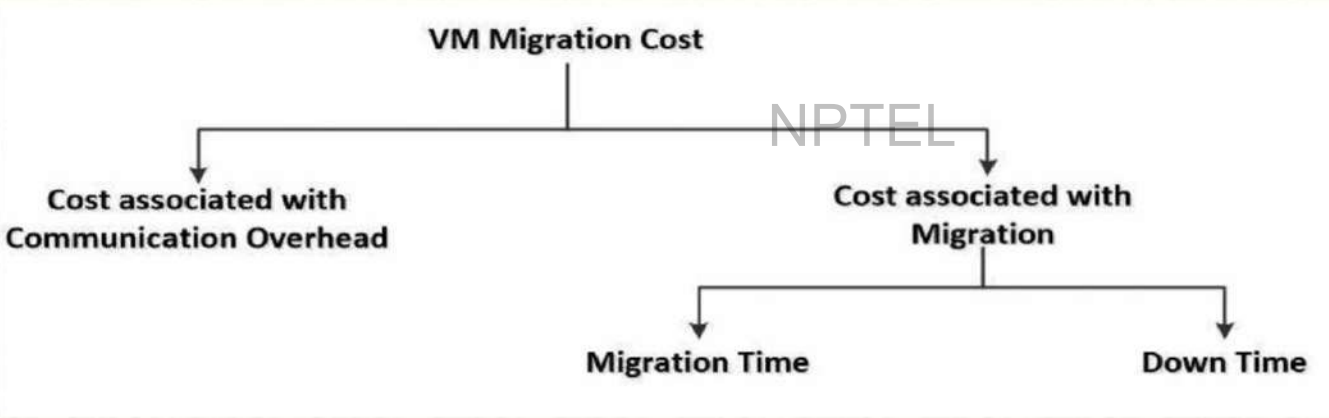
- **Hot or Live Migration:** In case of a hot or live migration the VM executing on the source machine continues to provide service during the migration process. In fact the target VM is not suspended during the migration process.

Live VM Migration

- Migrate an entire VM from one physical host to another
  - All user processes and kernel state
  - Without having to shut down the machine
- In case of non live migration the VM providing the services remains suspended during the entire migration process. Hence for large sized VMs the service downtime might be very high.
- For real time applications non live migration can cause severe degradation in service quality which is not tolerable.
- Two main approaches: Pre-copy and Post-copy

When to Migrate?

- To remove a physical machine from service.
- To relieve load on congested hosts.

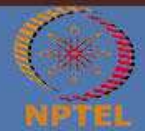


Migration - Concerns

- Minimize the downtime
  - Downtime refers to the total amount of time services remain unavailable to the users.
- Minimize total migration time
  - Migration time refers to the total time taken to move a VM from the source host to the destination host. It can be considered as the total time taken for the entire migration process.
- Migration does not unnecessarily disrupt active services through resource contention (e.g., CPU, network bandwidth) with the migrating OS.

What to Migrate?

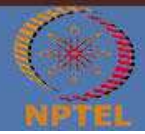
- CPU context of VM, contents of Main Memory
- Disk
  - If NAS (network attached storage) that is accessible from both hosts, or local disk is mirrored – migrating disk data may not be critical
- Network: *assume both hosts on same LAN*
  - Migrate IP address, advertise new MAC address to IP mapping via ARP reply
  - Migrate MAC address, let switches learn new MAC location
  - Network packets redirected to new location (with transient losses)
- I/O devices
  - Virtual I/O devices easier to migrate, direct device assignment of physical devices to VMs may be difficult to migrate



Memory Migration - Steps

- **Push** - Source VM continues running while certain pages are pushed across the network to the new destination. To ensure consistency, pages modified during this process must be re-sent.
- **Stop-and-copy** - Source VM is stopped, pages are copied across to the destination VM, then the new VM is started.
- **Pull** - The new VM executes and, if it accesses a page that has not yet been copied, this page is faulted in ("pulled") across the network from the source VM.

- Pure Stop-and-Copy
  - Simple but both downtime and total migration time are proportional to the amount of physical memory allocated to the VM.
  - May lead to an unacceptable outage if the VM is running a live service.



Live Migration - Phases

- **Pre-Copy Phase**: It is carried out over several rounds. The VM continues to execute at the source, while its memory is copied to the destination.
- **Pre-copy Termination Phase**: Stopping criteria of Pre-Copy phase takes one of the following thresholds into account: (i) The number of rounds exceeds a threshold. (ii) The total memory transmitted exceeds a threshold. (iii) The number of dirtied pages in the previous round drops below a threshold.
- **Stop-and-Copy Phase**: In this phase, execution of the VM to be migrated is suspended at the source. Then, the remaining dirty pages and, state of the CPU is copied to the destination host, where the execution of VM is resumed.

Iterative Pre Copy Live Memory Migration

- **Pre- copy Phase**:
  - This phase may be carried out over several rounds.
  - The VM continues to execute at the source host, while its memory is copied to the destination host.
  - Active pages of the VM to be migrated are copied iteratively in each round.
  - During the copying process some active page might get dirtied at the source host, which are again resent in the subsequent rounds to ensure memory consistency.
- **Pre copy-termination phase**: Stopping criteria - options
  - Number of rounds exceeds a threshold.
  - Total memory transmitted exceeds a threshold.
  - Number of dirtied pages in the previous round drops below a threshold.
- **Stop-and-Copy Phase**:
  - In this phase, execution of the VM to be migrated is suspended at the source.
  - Then, the remaining dirty pages and, state of the CPU is copied to the destination host, where the execution of VM is resumed.
- **Restarting Phase**: Restart the VM on destination server.

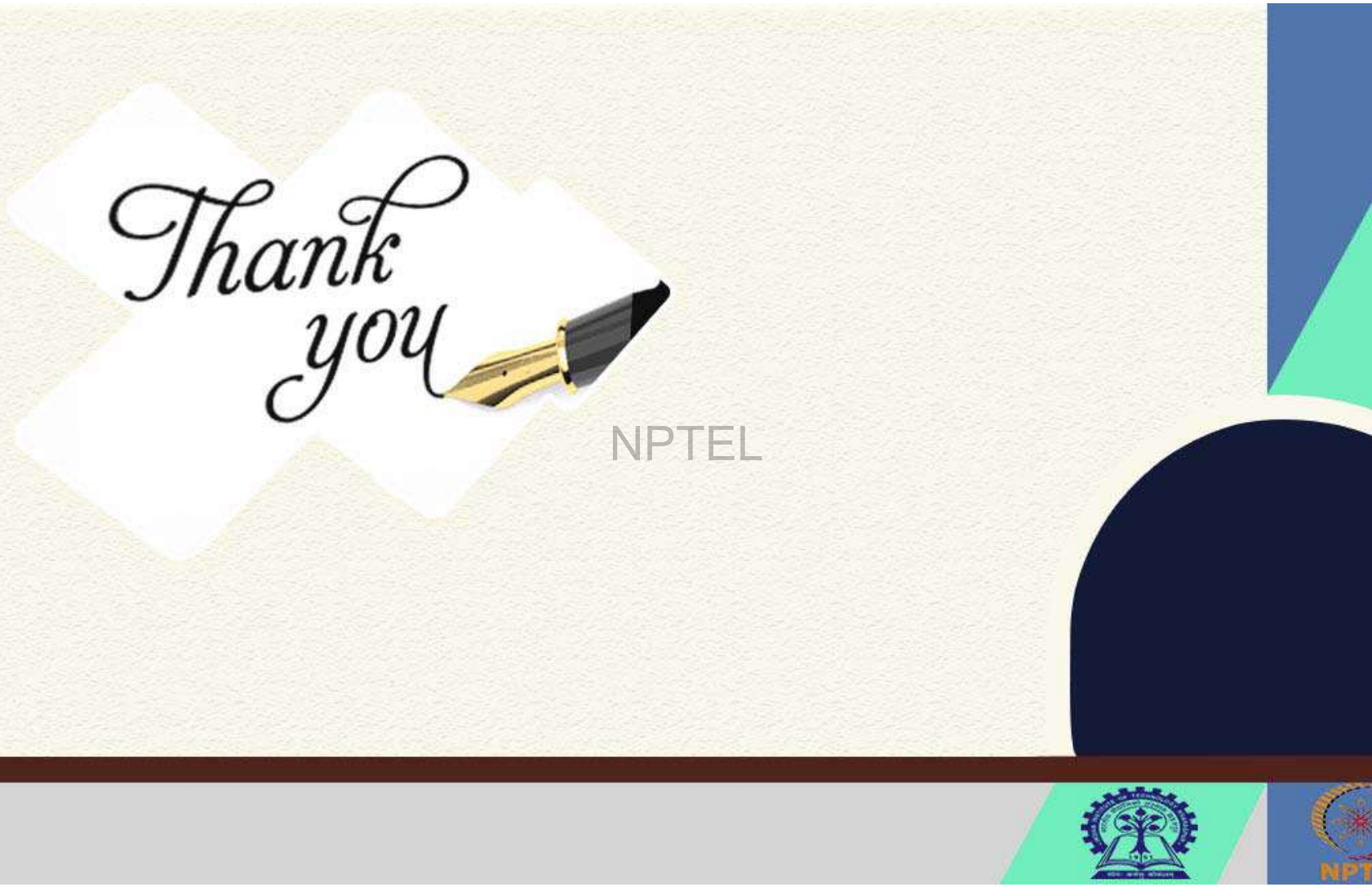
Post-copy Live Memory Migration

- **Stop Phase**: Stop the source VM and copy the CPU state to the destination VM.
- **Restart Phase**: Restart the destination VM.
- **On-demand Copy:** Copy the VM memory according to the demand.

➢ *In the post-copy strategy, when the VM is restarted, the VM memory is empty. If the VM tries to access a memory page that has not yet been copied, this memory page needs to be brought from the source VM. However, most of the time, some memory pages will not be used, so we only need to copy the VM memory according to the demand.*

REFERENCES

- Kai Hwang, Geoffrey C. Fox, Jack J. Dongarra, Distributed and Cloud Computing - From Parallel Processing to the Internet of Things, Morgan Kaufmann, Elsevier, 2012
- Christian Limpach, Ian Pratt, Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Andrew Warfield, Live Migration of Virtual Machines
- Michael R. Hines and Kartik Gopalan, Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning


Thank you
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Cloud Computing

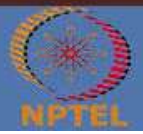
Prof. Soumya K Ghosh

Department of Computer Science and Engineering

Module 10: Cloud Computing Paradigm
Lecture 46: Cloud Migration - II

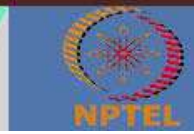
CONCEPTS COVERED

- VM Migration - Basics
- Migration strategies



KEYWORDS

- Virtual Machine (VM)
- VM Migration

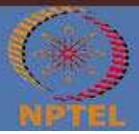


VM Migration (contd.)

NPTEL

VM Migration

- VM Migration – It is process to move running applications or VMs from one physical server/ host to another host.
- Processor state, storage, memory and network connection are moved from one host to another host
- Why to migrate VMs?
  - Distribute VM load efficiently across servers in a cloud
  - System maintenance



VM Live Migration – Requirements

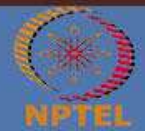
- **Load Balancing**: When the load is considerably unbalanced and impending downtime often require simultaneous VM (s) migration.
- **Fault tolerance**: Fault is another challenge to guarantee the critical service availability and reliability. Failures should be anticipated and proactively handled.
- **Power management**: Switching the idle mode server to either sleep mode or off mode based on resource demands, that leads to energy saving. VM live migration is a good technique for cloud power efficiency.
- **Resource sharing**: Challenge of limited hardware resources like memory, cache, and CPU cycles can be solved by relocating VM's from over-loaded server to under-loaded server.
- **System maintenance**: Physical system required to be upgraded and serviced, so VMs of that physical server must be migrated to an alternate server so that services are available to users without interruption

Live Migration – Pre-copy Approach

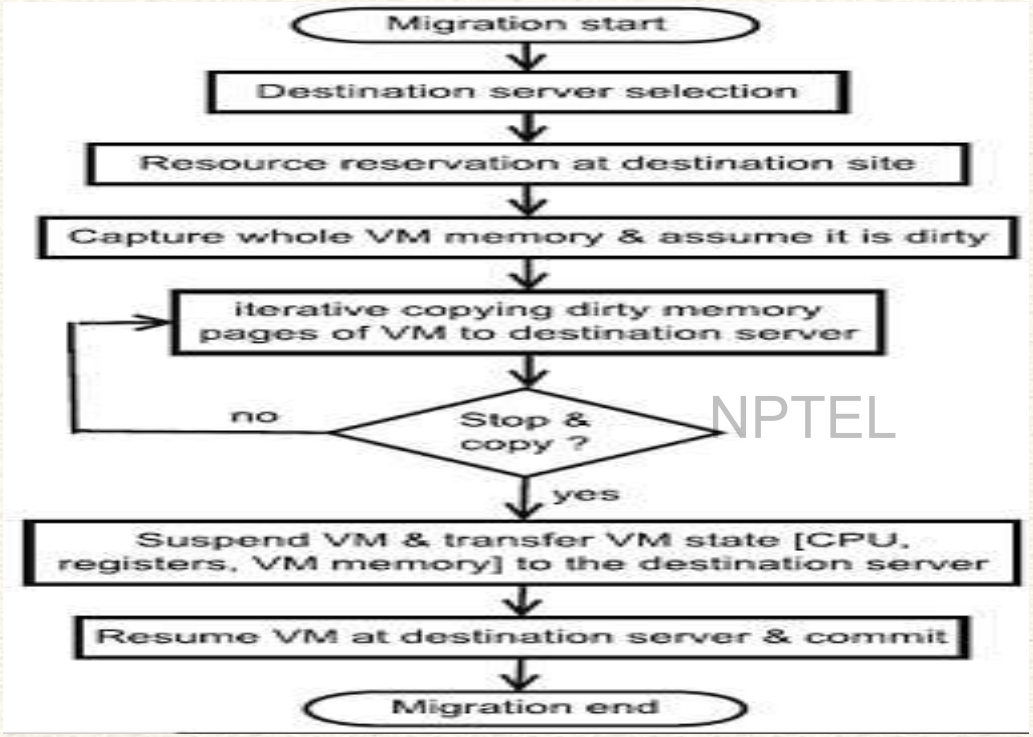
- Uses iterative push phase that is followed by stop-and-copy phase.
- Because of iterative procedure, some memory pages have been updated/modified, called dirty pages are regenerated on the source server during migration iterations.
- Dirty pages resend to the destination host in a future iteration, hence some of the or frequently access memory pages are sent several times. It causes long migration time.
- In the **first phase**, all pages are transferred while VM running continuously on the source host. Further round(s), dirty pages are re-sent.

Live Migration – Pre-copy Approach (contd.)

- **Second phase** is termination phase which depends on the defined threshold. The termination is executed if any one out of three conditions: (i) the number of iterations exceeds pre-defined iterations, or (ii) the total amount of memory that has been sent or (iii) the number of dirty pages in just previous round fall below the defined threshold.
- In the last, **stops-and-copy phase**, migrating VM is suspended at source server, after that move processors state and remaining dirty pages.
- When VM migration process is completed in the correct way then hypervisor resumes migrant VM on the destination server.
- KVM, Xen, and VMware hypervisor use the pre-copy technique for live VM migration.



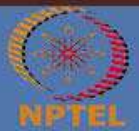
Live Migration – Pre-copy Approach



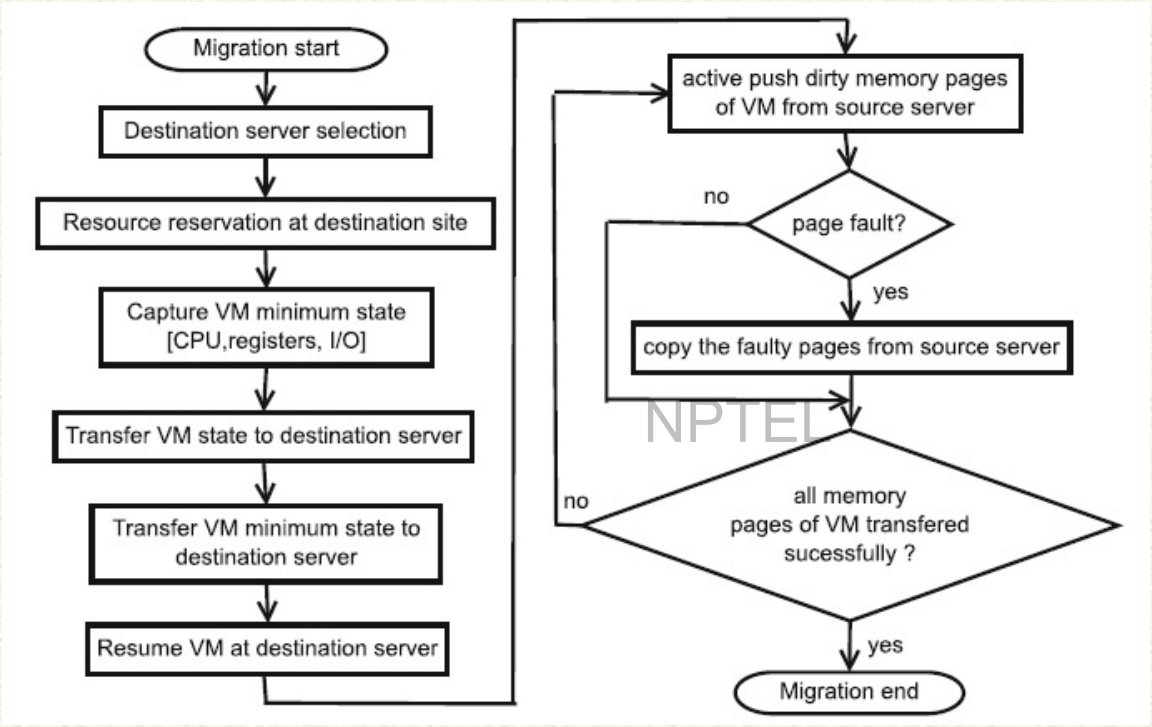
Live Migration – Post-copy Approach

- In post-copy migration technique, processor state transfer before memory content and then
- VM could be started at the destination server.
- Post-copy VM migration technique investigates demand paging, active push, pre-paging etc. approaches for prefetching of memory pages at the destination server.

- ✓ **Stop Phase**: Stop the source VM and copy the CPU state to the destination VM.
- ✓ **Restart Phase**: Restart the destination VM.
- ✓ **On-demand Copy:** Copy the VM memory according to the demand.



Live Migration – Post-copy Approach



VM Migration – Analysis

- Let T_{mig} be the total migration time.
- Let T_{down} be the total down time.
- For non-live migration of a single VM the migration time T_{mig} can be calculated as follows:

$$T_{mig} = V_m / R \ .$$

where, V_m is the size i.e. memory of the VM and R is the transmission rate.

- In non-live migration, down time is same as the migration time because the services of the VM is suspended during the entire migration process.

$$T_{down} = T_{mig}$$

Note: Transmission rate remains fixed for the entire duration of migration.

VM Migration – Analysis (contd.)

- Let **n** represent the total number of iterations in the pre copy cycle.
- Let $\mathbf{T_{i,j}}$ represents the total time that the $\mathbf{j^{th}}$ iterative transmits the $\mathbf{i^{th}}$ virtual machine's memory.
- V_m : the memory of a VM.
- V_{th} : threshold for stopping the iterations.
- n_{max} : maximum number of iterations.
- r = (P*D)/R.

  where P is page size and D is the dirtying rate, R is the transmission rate.
- T_{res} denotes the time taken to restart the VM on the destination server.

VM Migration – Analysis (contd.)

- Pre-copy migration mechanism: the VMs memory can be migrated iteratively.
- We can compute the total migration time $T_{i,mig}$ of the i^{th} VM as follows.
- $$T_{i,mig} = \sum_{j=0}^{n}(T_{i,j}) = \frac{V_m}{R}\left(\frac{1-r^{n+1}}{1-r}\right) + T_{res}$$
- $$T_{i,down} = r^n\left(\frac{V_m}{R}\right) + T_{res}$$

VM Migration – Analysis (contd.)

- Round 0 : $t_0 = \frac{V_m}{R}$
- Round 1: $t_1 = \frac{(P*D)}{R} * t_0 = \frac{(P*D)}{R} * \frac{V_m}{R} = r*(\frac{V_m}{R})$
- Round 2: $t_2 = \frac{(P*D)}{R} * t_1 = \frac{(P*D)}{R} * (r*\frac{V_m}{R}) = r^2(\frac{V_m}{R})$
- Round 3: $t_3 = \frac{(P*D)}{R} * t_2 = \frac{(P*D)}{R} * (r^2\frac{V_m}{R}) = r^3(\frac{V_m}{R})$.

…….. .

- Round **n–1**: $t_{n-1} = \frac{(P*D)}{R} * t_{n-2} = \frac{(P*D)}{R} * (r^{n-2} * \frac{V_m}{R}) = r^{n-1} * (\frac{V_m}{R})$
- Round **n** (Stop and Copy): $t_n = \frac{(P*D)}{R} * t_{n-1} = \frac{(P*D)}{R} (r^{n-1} * \frac{V_m}{R}) = r^n(\frac{V_m}{R})$
- $T = t_0+t_1+\ldots+t_{n-1} + t_n = \frac{V_m}{R}(1+r+r^2+r^3+\ldots+r^{n-1}+r^n) = \frac{V_m}{R}(\frac{1-r^{n+1}}{1-r})$.

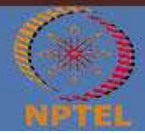
VM Migration – Analysis (contd.)

Estimation of Number of Rounds (n)

- Volume of dirty data to be transferred in round *j* : $r^j.V_m$
- $r^j.V_m < V_{th}$
- => $j = \lceil log_r \frac{V_{th}}{V_m} \rceil$
- n= min($\lceil log_r \frac{V_{th}}{V_m} \rceil$,$n_{max}$)

Multiple VMs Migration

- Generally multiple VMs are migrated from a source host to the destination host.
- Typical strategies for migration multiple VMs:
  - Serial Migration.
  - Parallel migration.



Serial Migration

In case of serial migration of ‘m’ correlated VMs of same type the procedure is as follows:

- The first VM that is selected to be migrated executes its pre- copy cycle and the other (m-1) VMs continue to provide services.
- As soon as the first VM enters into the stop and copy phase the remaining (m-1) VMs are suspended and are copied after the first VM completes its stop and copy phase.
- Reason for stopping the remaining (m-1) VMs is to stop those VMs from dirtying memory.
- Assumption: each VM that is copied at full transmission rate (R).
- Downtime for the serial migration includes the stop and copy phase of the first VM, the migration time for the (m-1) VMs and the time to resume the VMs at the destination host.

Serial Migration

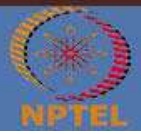
- Consider there are ‘m’ VMs that are to be migrated serially.
- Migration time and downtime for serial migration strategy can be calculated as follows:

$$T^{s}_{mig} = \sum_{i=1}^{m}(T_{i,mig}) = \frac{m.Vm}{R}\left(\frac{1-r^{n+1}}{1-r}\right) + T_{res}$$

$$T^{s}_{down} = \frac{V_m}{R}.\, r^{n} + (m-1)\frac{V_m}{R}\left(\frac{1-r^{n+1}}{1-r}\right) + T_{res}$$

Parallel Migration

- Major difference between parallel and serial migrations is that all ‘m’ VMs start their pre-copy cycles simultaneously.
- In fact each VM shares (R/m) of the transmission capacity.
- As the VM sizes are same and transmission rates are same the VMs begin the stop and copy phase as the same time and they end the stop and copy phase also at the same time.
- Since the stop and copy phase is executed in parallel and they consume the same amount of time the downtime is in fact equivalent to the time taken by the stop and copy phase for any VM added to the time taken to resume the VMs at the destination host.

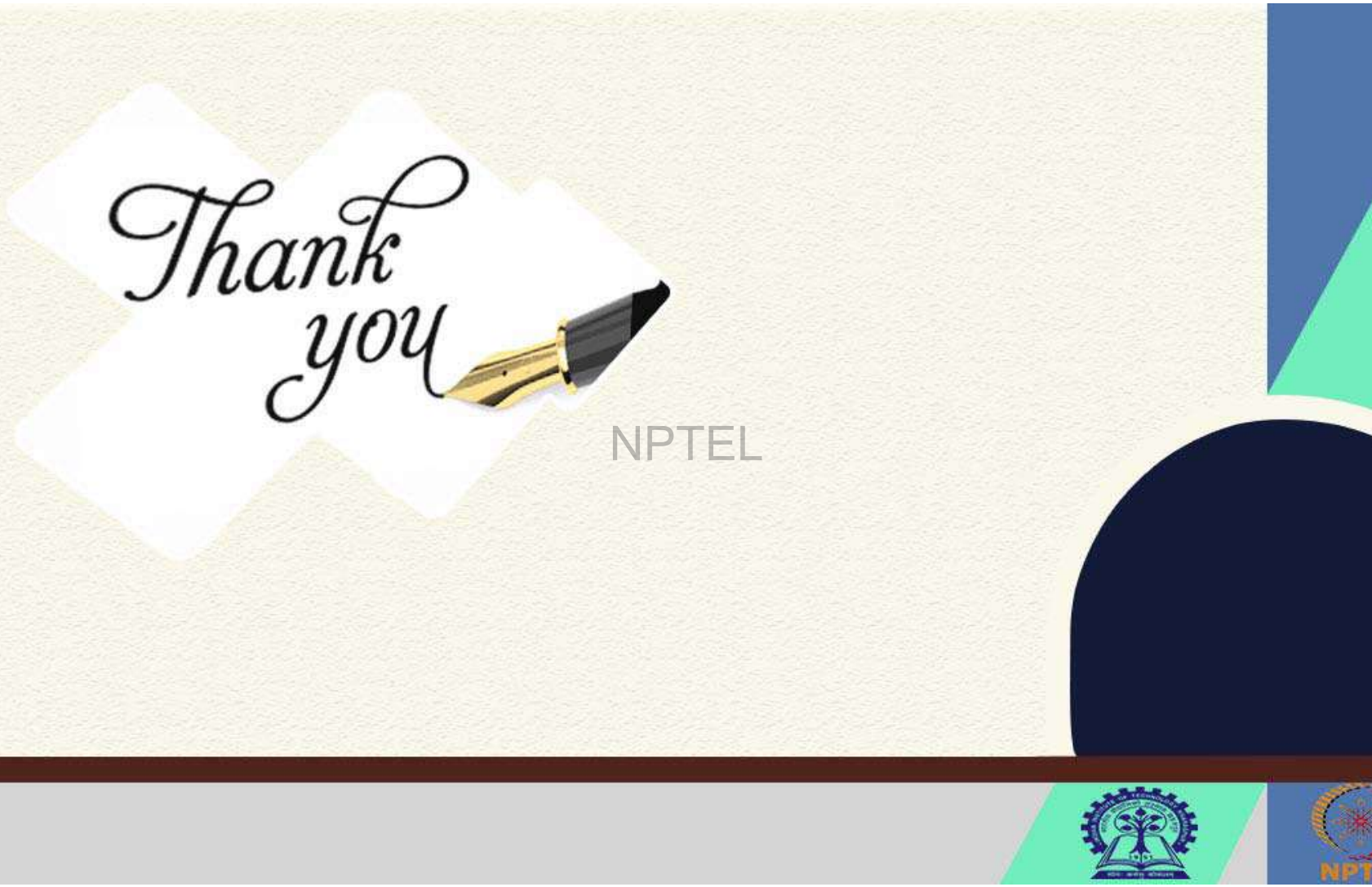


Parallel Migration

- $T^{p}_{mig} = \sum_{i=1}^{m}(T_{i,mig}) = \frac{m.Vm}{R}\left(\frac{1-mr^{n(p)+1}}{1-mr}\right)+T_{res}$
- $T^{p}_{down} = \frac{m.Vm}{R}.\ (m.r)^{n(p)} + T_{res}$

REFERENCES

- Kai Hwang, Geoffrey C. Fox, Jack J. Dongarra, Distributed and Cloud Computing - From Parallel Processing to the Internet of Things, Morgan Kaufmann, Elsevier, 2012
- Christian Limpach, Ian Pratt, Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Andrew Warfield, Live Migration of Virtual Machines, NSDI, 2005
- Michael R. Hines and Kartik Gopalan, Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning, 2009
- Anita Choudhary, Mahesh Chandra Govil, Girdhari Singh, Lalit K. Awasthi, Emmanuel S. Pilli, Divya Kapil, A critical survey of live virtual machine migration techniques, Journal of Cloud Computing, Springer, 6(23), 2017


Thank you
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Cloud Computing

Prof. Soumya K Ghosh

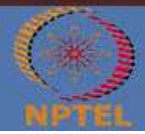
Department of Computer Science and Engineering

Module 10: Cloud Computing Paradigm

Lecture 47: Container based Virtualization - I

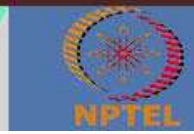
CONCEPTS COVERED

- Containers
- Container based Virtualization
- Kubernetes
- Docker Container



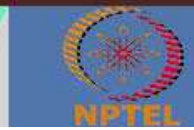
KEYWORDS

- Container
- Virtualization



Containers

NPTEL



Containers - Introduction

- Virtualization helps to share resources among many customers in cloud computing.
- Container is a lightweight virtualization technique.
- Container packages the code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
- *Docker* is an open platform for developing, shipping, and running applications.
- *Kubernetes* is an open-source system for automating deployment, scaling, and management of containerized applications.

Containers - Introduction

- Containers are packages of software that contain all of the necessary elements to run in any environment.
- Containers virtualize the operating system and run anywhere, from a private data center to the public cloud or even on a developer's personal laptop.
- *Containers are lightweight packages of the application code together with dependencies such as specific versions of programming language runtimes and libraries required to run the software services.*
- Containers make it easy to share CPU, memory, storage, and network resources at the operating systems level and offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run.

Ref: https://cloud.google.com/learn/what-are-containers

Containers - Needs

- Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run.
- This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer's personal laptop.
- *Agile development*: Containers allow the developers to move much more quickly by avoiding concerns about dependencies and environments.
- *Efficient operations*: Containers are lightweight and allow to use just the computing resources one need – thus running the applications efficiently.
- *Run anywhere*: Containers are able to run virtually anywhere. .

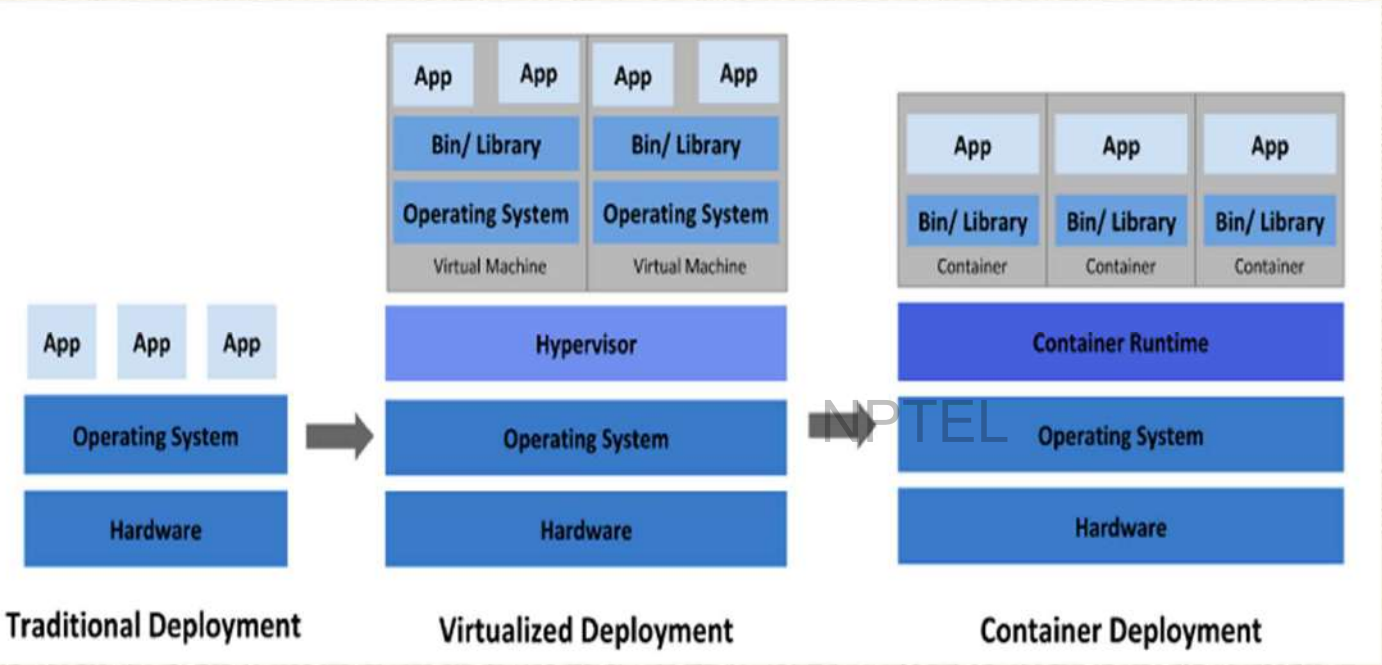
Ref: https://cloud.google.com/learn/what-are-containers

Containers – Major Benefits

- **Separation of responsibility**: Containerization provides a clear separation of responsibility, as developers focus on application logic and dependencies, while IT operations teams can focus on deployment and management instead of application details such as specific software versions and configurations.
- **Workload portability**: Containers can run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or on physical servers; on a developer's machine or in data centers on-premises; and of course, in the public cloud.
- **Application isolation**: Containers virtualize CPU, memory, storage, and network resources at the operating system level, providing developers with a view of the OS logically isolated from other applications.

Ref: https://cloud.google.com/learn/what-are-containers

Application Deployment



Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

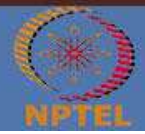
Traditional – Virtualized – Container Deployments

- **Traditional deployment :** Applications run on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.
- **Virtualized deployment :** Allows to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs. It allows better utilization of resources in a physical server and allows better scalability. Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.
- **Container deployment:** Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. A container has its own filesystem, share of CPU, memory, process space, and more. As containers are decoupled from the underlying infrastructure, they are portable across clouds and different OS distributions.

Containers and VMs

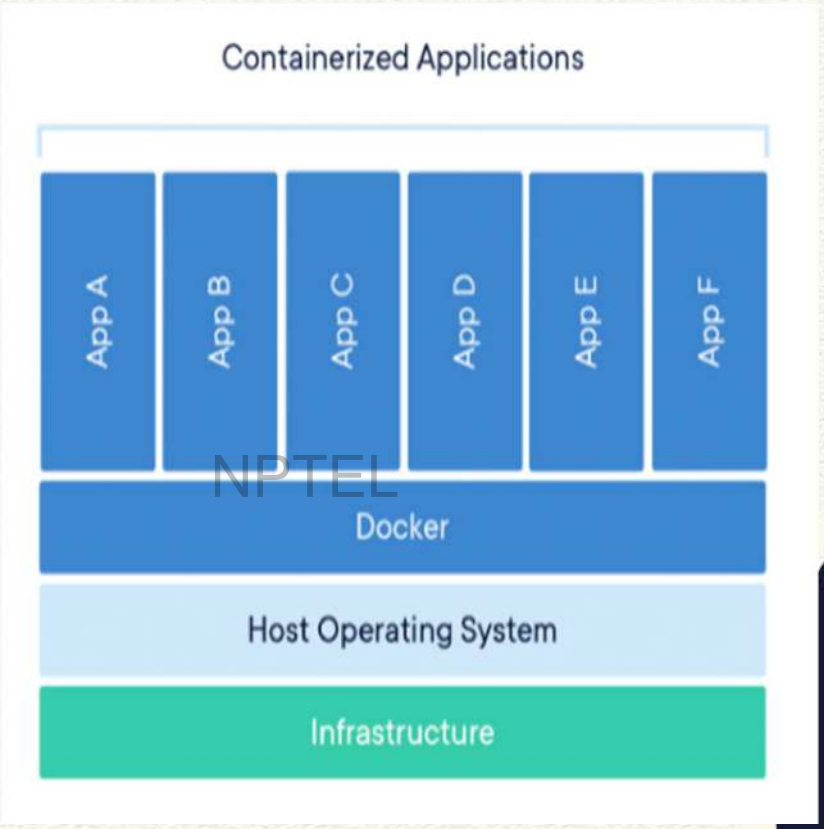
- VMs: a guest operating system such as Linux or Windows runs on top of a host operating system with access to the underlying hardware.
- Containers are often compared to virtual machines (VMs). Like virtual machines, containers allow one to package the application together with libraries and other dependencies, providing isolated environments for running your software services.
- However, the containers offer a far more lightweight unit for developers and IT Ops teams to work with, carrying a myriad of benefits.

- Containers are much more lightweight than VMs
- Containers virtualize at the OS level while VMs virtualize at the hardware level
- Containers share the OS kernel and use a fraction of the memory VMs require



Container

- A container is a sandboxed process that is isolated from all other processes on the host machine.
- A container is a runnable instance of an image.
- One can create, start, stop, move, or delete a container using the API (e.g. DockerAPI) or CLI.
- A container can be run on local machines, virtual machines or deployed to the cloud.

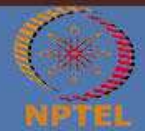


Ref: https://www.docker.com/resources/what-container

Kubernetes

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
- The name Kubernetes originates from Greek, meaning helmsman or pilot.
- Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions.
- However, Kubernetes is not monolithic, and these default solutions are optional and pluggable.
- Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

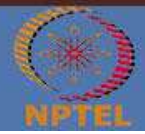
Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/



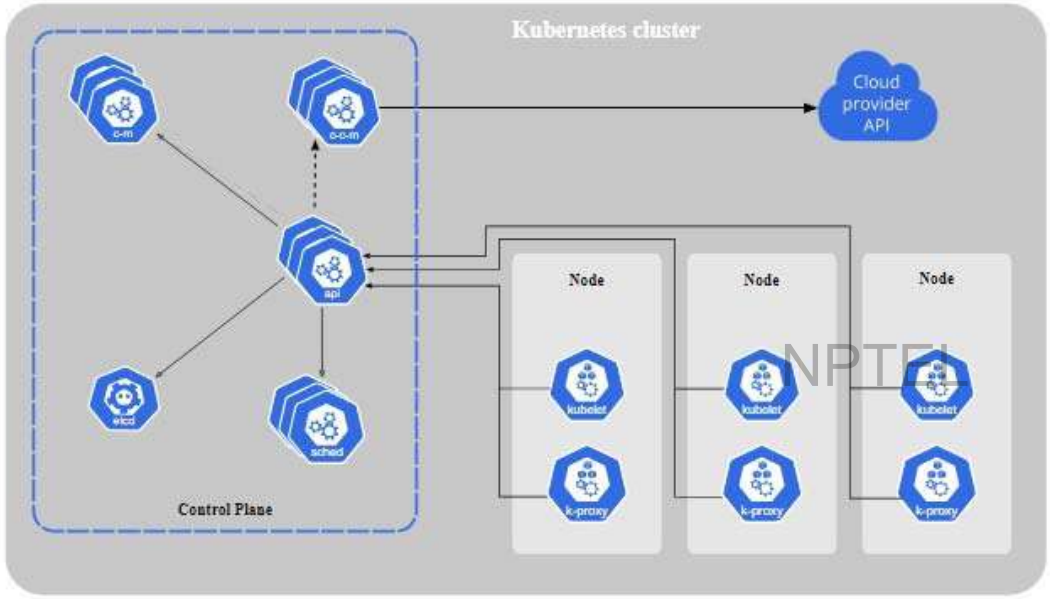
Kubernetes Components

- A Kubernetes cluster consists of a set of worker machines, called **nodes**, that run containerized applications. Every cluster has at least one worker node.
- The worker node(s) host the **Pods** that are the components of the application workload.
- The **control plane** manages the worker nodes and the Pods in the cluster.
- In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/



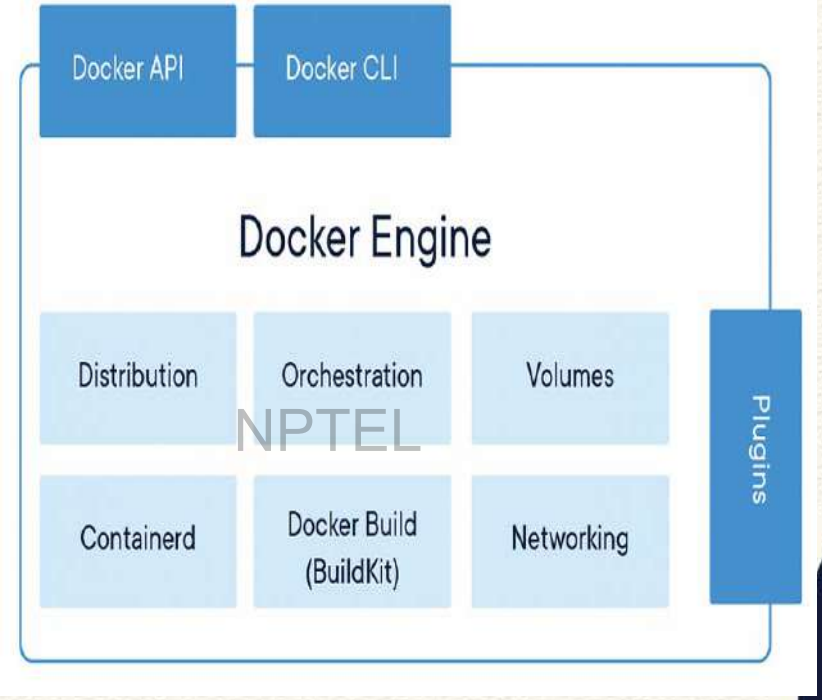
Kubernetes Cluster Components



Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Docker Engine

- Docker containers that run on Docker Engine:
- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

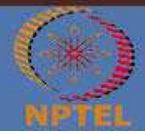


Ref: https://www.docker.com/resources/what-container

Dockers

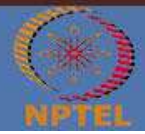
- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
- Container images become containers at runtime and in the case of Docker containers - images become containers when they run on **Docker Engine**.
- Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

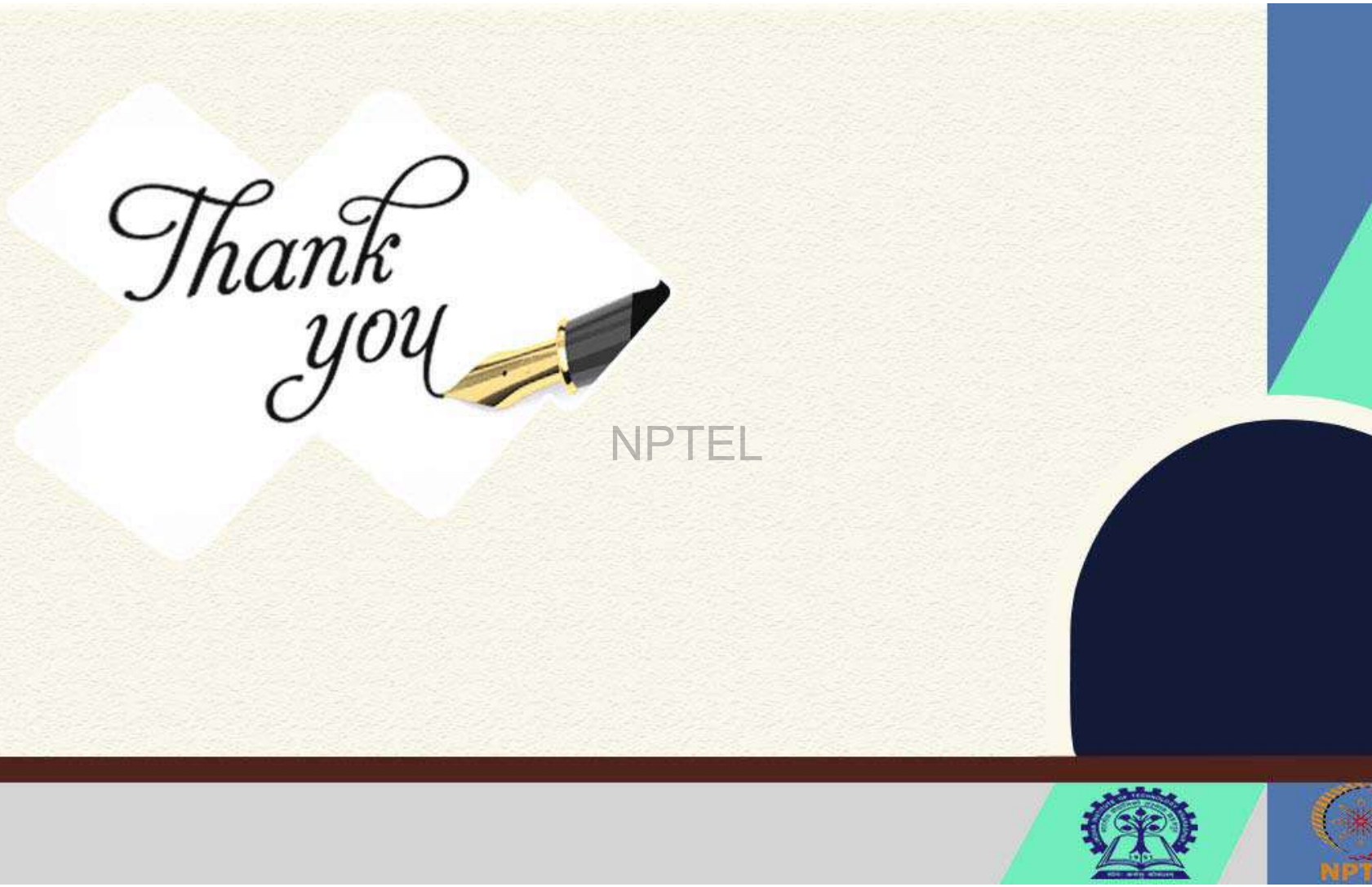
Ref: https://www.docker.com/resources/what-container



REFERENCES

- https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
- https://www.docker.com/resources/what-container
- https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler




Thank you
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Cloud Computing

Prof. Soumya K Ghosh

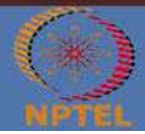
Department of Computer Science and Engineering

Module 10: Cloud Computing Paradigm

Lecture 48: Container – II (Docker)

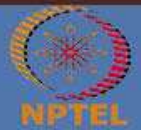
CONCEPTS COVERED

- Docker Container – Overview
- Docker – Components
- Docker – Architecture

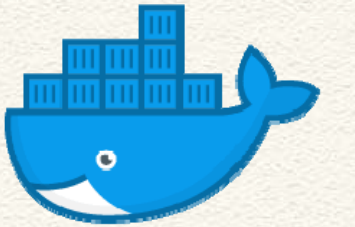


KEYWORDS

- Container
- Docker Container



Docker

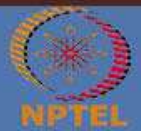


https://www.docker.com/

Docker - Overview

- Docker is a platform that allows you to “build, ship, and run any app, anywhere.”
- Considered to be a standard way of solving one of the challenging aspects of software: deployment.
- Traditionally, the development pipeline typically involved combinations of various technologies for managing the movement of software, such as virtual machines, configuration management tools, package management systems, and complex webs of library dependencies.
  - All these tools needed to be managed and maintained by specialist engineers, and most had their own unique ways of being configured.

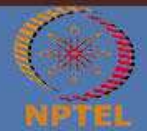
Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808



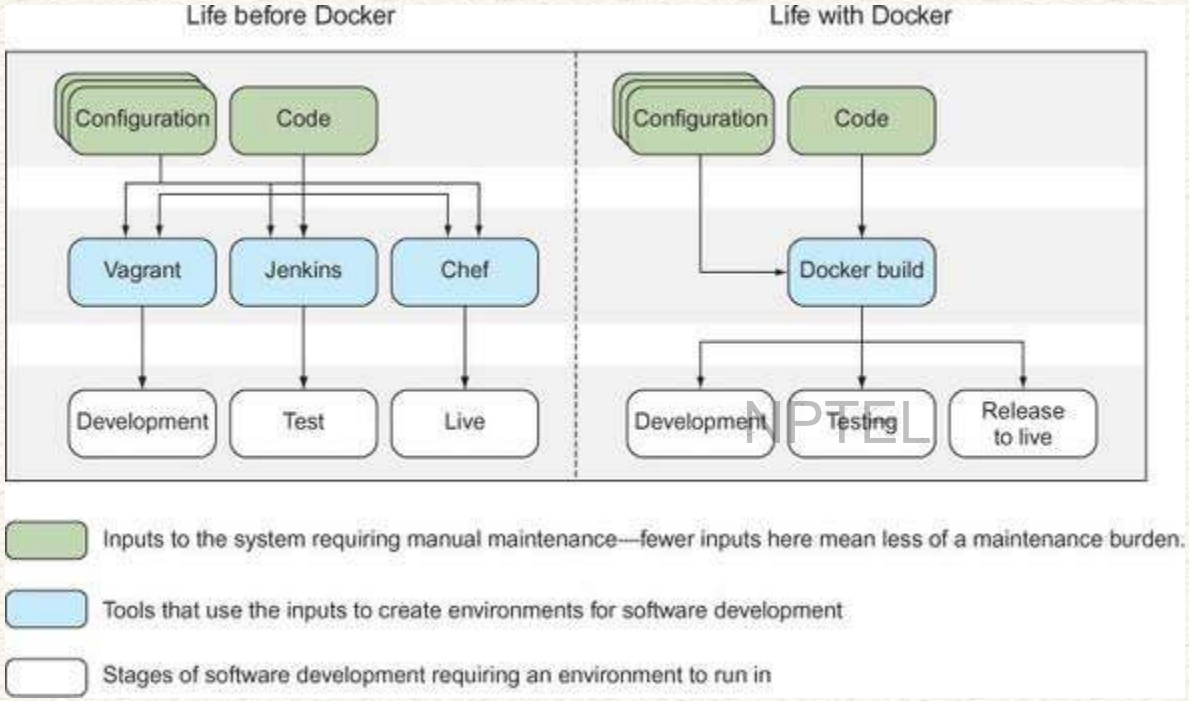
Docker - Overview

- Docker has changed the traditional approach - Everything goes through a common pipeline to a single output that can be used on any target—there's no need to continue maintaining a bewildering array of tool configurations
- At the same time, there's no need to throw away the existing software stack if it works for you—you can package it up in a Docker container as-is, for others to consume.
- Addionally, you can see how these containers were built, so if you need to dig into the details, you can.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808



Docker – Big Picture

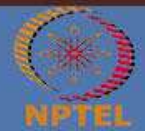


Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Analogy

- *Analogy:* Tradionally, a docker was a laborer who moved commercial goods into and out of ships when they docked at ports. There were boxes and items of differing sizes and shapes, and experienced dockers were prized for their ability to fit goods into ships by hand in cost-effective ways. Hiring people to move stuff around wasn't cheap, but there was no alternative!
- This may sound familiar to anyone working in software. Much time and intellectual energy is spent getting metaphorically odd-shaped software into differently-sized metaphorical ships full of other odd-shaped software, so they can be sold to users or businesses elsewhere.

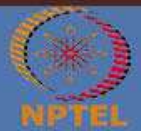
Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808



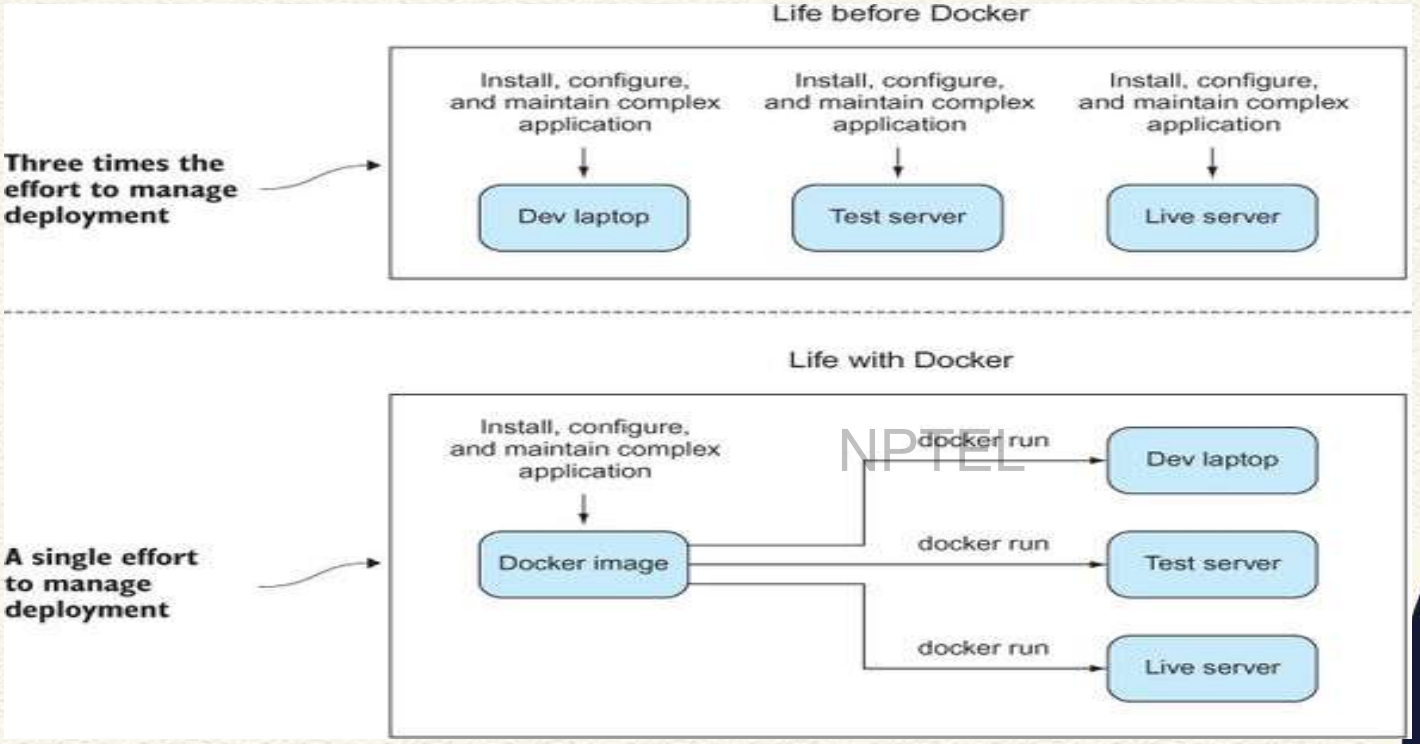
Docker - Benefit

- Before Docker, deploying software to different environments required significant effort. Even if you weren't hand-running scripts to provision software on different machines (and plenty of people do exactly that), you'd still have to handle the configuration management tools that manage state on what are increasingly fast-moving environments starved of resources.
- Even when these efforts were encapsulated in VMs, a lot of time was spent managing the deployment of these VMs, waiting for them to boot, and managing the overhead of resource use they created.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808



Docker - Benefit



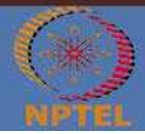
Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Benefit

- With Docker, the configuration effort is separated from the resource management, and the deployment effort is trivial:
  - run docker, and the environment's image is pulled down and ready to run, consuming fewer resources and contained so that it doesn't interfere with other environments.

- You don't need to worry about whether your container is going to be shipped to a Red Hat machine, an Ubuntu machine, or a CentOS VM image; as long as it has Docker on it, it will run

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808



Docker - Advantage

- **Replacing virtual machines (VMs)**: Docker can be used to replace VMs in many situations. If you only care about the application, not the operating system, Docker can replace the VM.
  - Not only is Docker quicker than a VM to spin up, it's more lightweight to move around, and due to its layered filesystem, you can more easily and quickly share changes with others. It's also rooted in the command line and is scriptable.

- **Prototyping software**: If you want to quickly experiment with software without either disrupting your existing setup or going through the hassle of provisioning a VM, Docker can give you a sandbox environment almost instantly. .

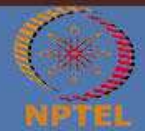
Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Advantage

- **Packaging software**: Because a Docker image has effectively no dependencies, it's a great way to package software. You can build your image and be sure that it can run on any modern Linux machine—think Java, without the need for a JVM.

- **Enabling a Microservices architecture**: Docker facilitates the decomposition of a complex system to a series of composable parts, which allows you to reason about your services in a more discrete way. This can allow you to restructure your software to make its parts more manageable and pluggable without affecting the whole.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808



Docker - Advantage

- **Modeling networks:** Several hundreds (even thousands) of isolated containers can be initiated on one machine, modeling a network can be done efficiently. .

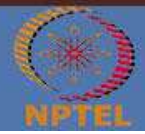
- **Enabling full-stack productivity when offline** - All the parts of the system can be bundled into Docker containers, you can orchestrate these to run on your laptop and work on the move, even when offline.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker - Advantage

- **Reducing debugging overhead**: Complex negotiations between different teams about software delivered is a commonplace within the industry.
  - Docker allows you to state clearly (even in script form) the steps for debugging a problem on a system with known properties, making bug and environment reproduction a much simpler affair, and one normally separated from the host environment provided.
- **Documenting software dependencies**: By building your images in a structured way, ready to be moved to different environments, Docker forces you to document your software dependencies explicitly from a base starting point..

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

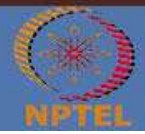


Docker - Advantage

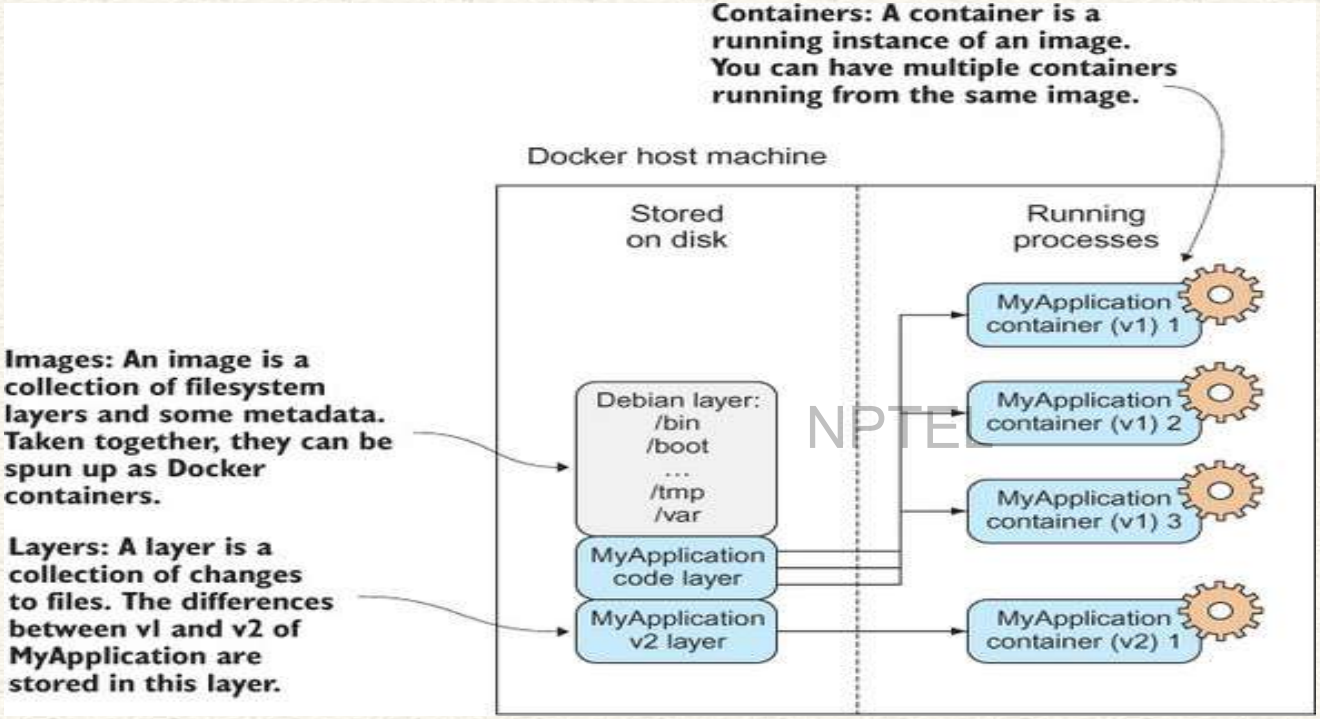
- **Enabling continuous delivery**: Continuous delivery (CD) is a paradigm for software delivery based on a pipeline that rebuilds the system on every change and then delivers to production (or "live") through an automated (or partly automated) process.

➢ Docker builds are more reproducible and replicable than traditional software building methods. This makes implementing Continuous delivery (CD) much easier.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808



Docker – Key Concepts

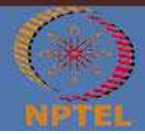


Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

Docker – Key Commands

Command	Purpose
`docker build`	Build a Docker image
`docker run`	Run a Docker image as a container
`docker commit`	Commit a Docker container as an image
`docker tag`	Tag a Docker image

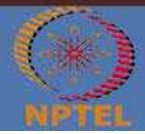
Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808



Docker – Architecture

- Docker on your host machine is split into two parts—a daemon with a RESTful API and a client that talks to the daemon.
- The private Docker registry is a service that stores Docker images. These can be requested from any Docker daemon that has the relevant access. This registry is on an internal network and isn't publicly accessible, so it's considered private.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

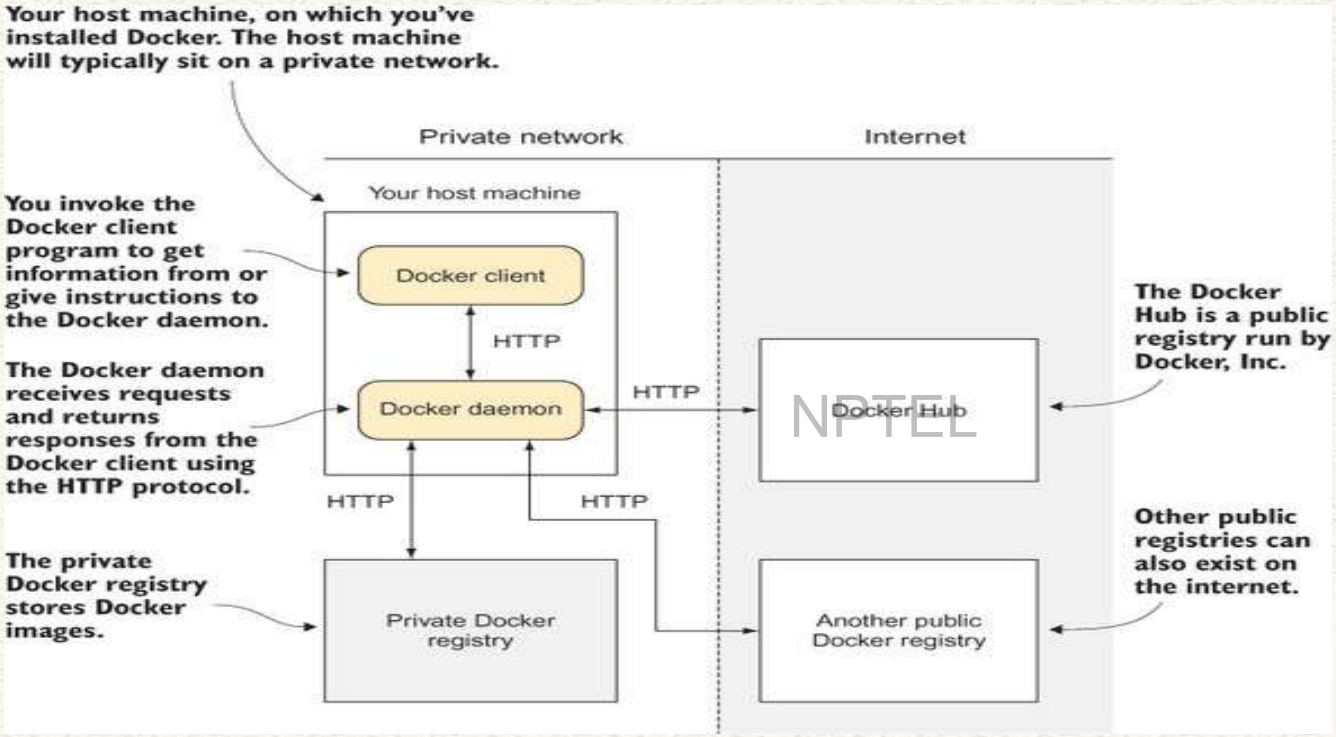


Docker – Architecture

- One invokes the Docker client to get information from or give instructions to the daemon; the daemon is a server that receives requests and returns responses from the client using the HTTP protocol.
- In turn, it will make requests to other services to send and receive images, also using the HTTP protocol.
- The server will accept requests from the command-line client or anyone else authorized to connect.
- The daemon is also responsible for taking care of your images and containers behind the scenes, whereas the client acts as the intermediary between you and the RESTful API.

Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

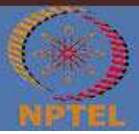
Docker – Architecture

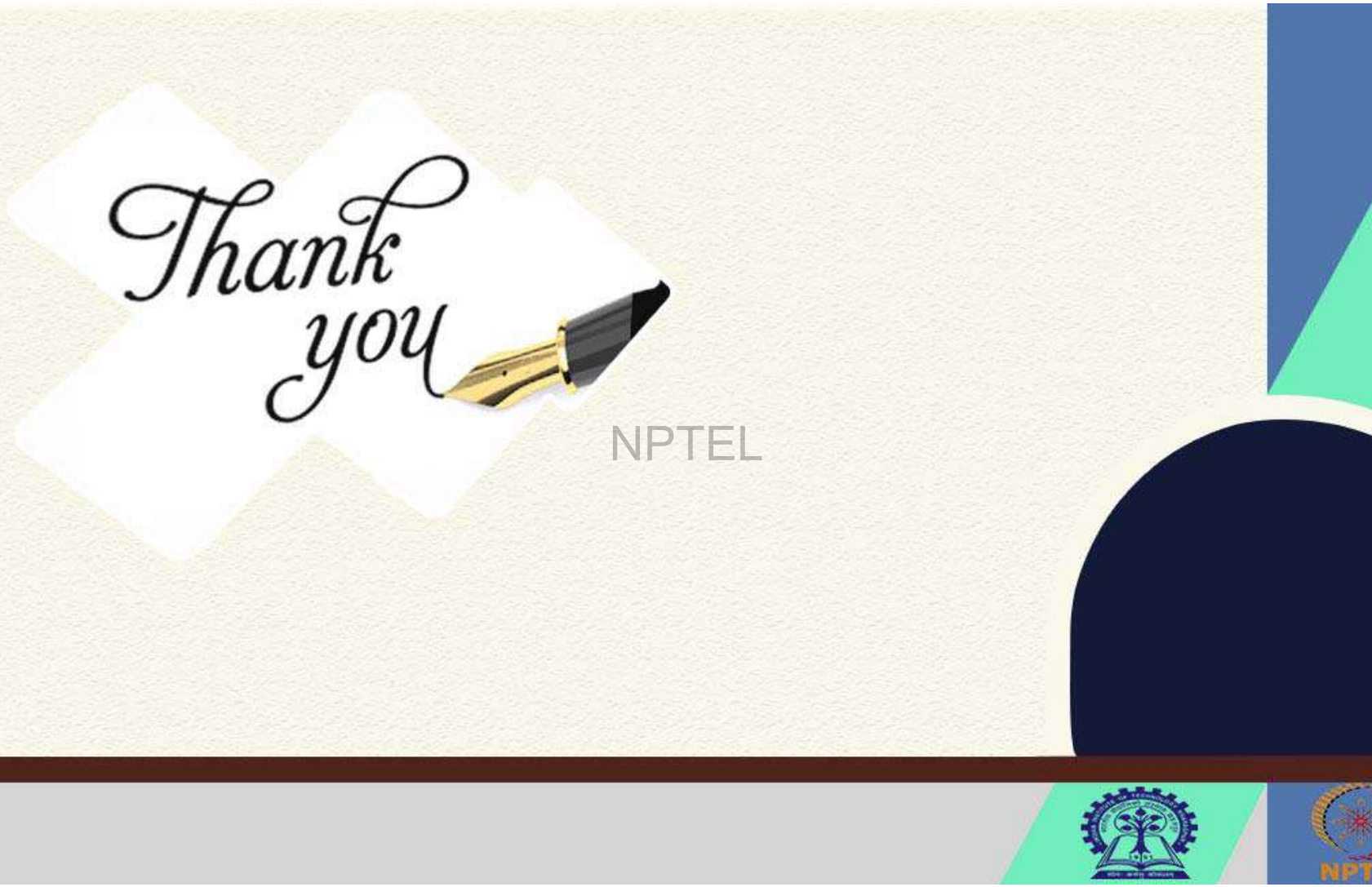


Ref: Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808

REFERENCES

- https://www.docker.com/
- Docker in Practice, Second Edition, Ian Miell and Aidan Hobson Sayers, February 2019, ISBN 9781617294808




Thank you
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Cloud Computing

Prof. Soumya K Ghosh

Department of Computer Science and Engineering

Module 10: Cloud Computing Paradigm

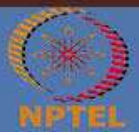
Lecture 49: Docker Container - Demo (Part-I)

CONCEPTS COVERED

- Docker Container - Demo

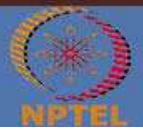
KEYWORDS

- Container
- Docker



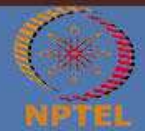
Docker Demo - I

NPTEL



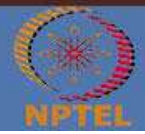
Introduction

- **Containers**
  - Standard unit of software
  - Packages up code and all its dependencies
  - Application runs quickly and reliably
- **Docker container image**
  - Lightweight
  - Standalone
  - Executable package of software
    - Includes everything needed to run an application



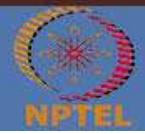
Demo - Objective

- MySQL and PHPMyAdmin on Docker platform
- MySQL
  - Widely used relational database package
  - Open-source
- PHPMyAdmin
  - A graphical user interface
  - Web-based
  - Connects to MySQL database
  - Widely used for managing MySQL databases



Standalone System (No Container)

- **Separate installation for**
  - MySQL
  - Web Server (Apache)
  - PHP
  - PHPMyAdmin
- **Transferring to other machine/ system**
  - Separate installation
  - Backup of data from old MySQL server
  - Restore the backup to new MySQL server

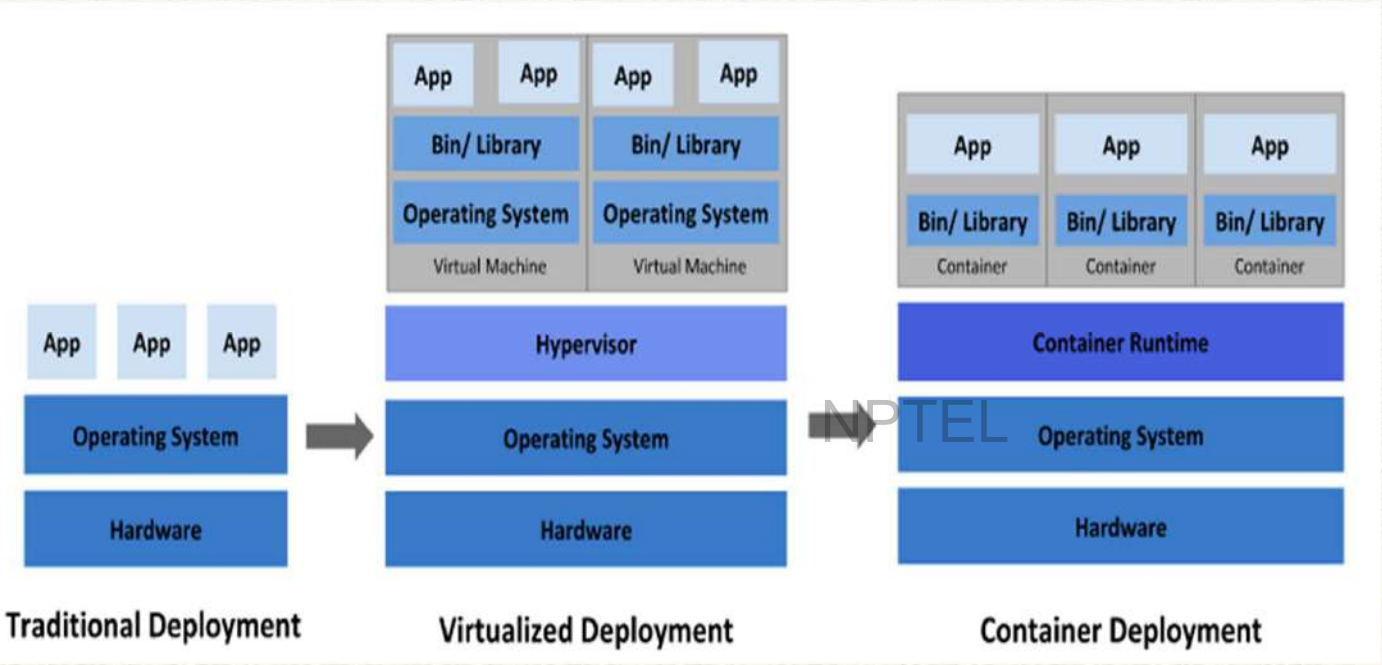


Containers – Major Benefits

- **Separation of responsibility**: Containerization provides a clear separation of responsibility, as developers focus on application logic and dependencies, while IT operations teams can focus on deployment and management instead of application details such as specific software versions and configurations.
- **Workload portability**: Containers can run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or on physical servers; on a developer's machine or in data centers on-premises; and of course, in the public cloud.
- **Application isolation**: Containers virtualize CPU, memory, storage, and network resources at the operating system level, providing developers with a view of the OS logically isolated from other applications.

Ref: https://cloud.google.com/learn/what-are-containers

Application Deployment



Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

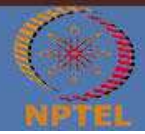
Traditional – Virtualized – Container Deployments

- **Traditional deployment :** Applications run on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.
- **Virtualized deployment :** Allows to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs. It allows better utilization of resources in a physical server and allows better scalability. Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.
- **Container deployment:** Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. A container has its own filesystem, share of CPU, memory, process space, and more. As containers are decoupled from the underlying infrastructure, they are portable across clouds and different OS distributions.

Containers and VMs

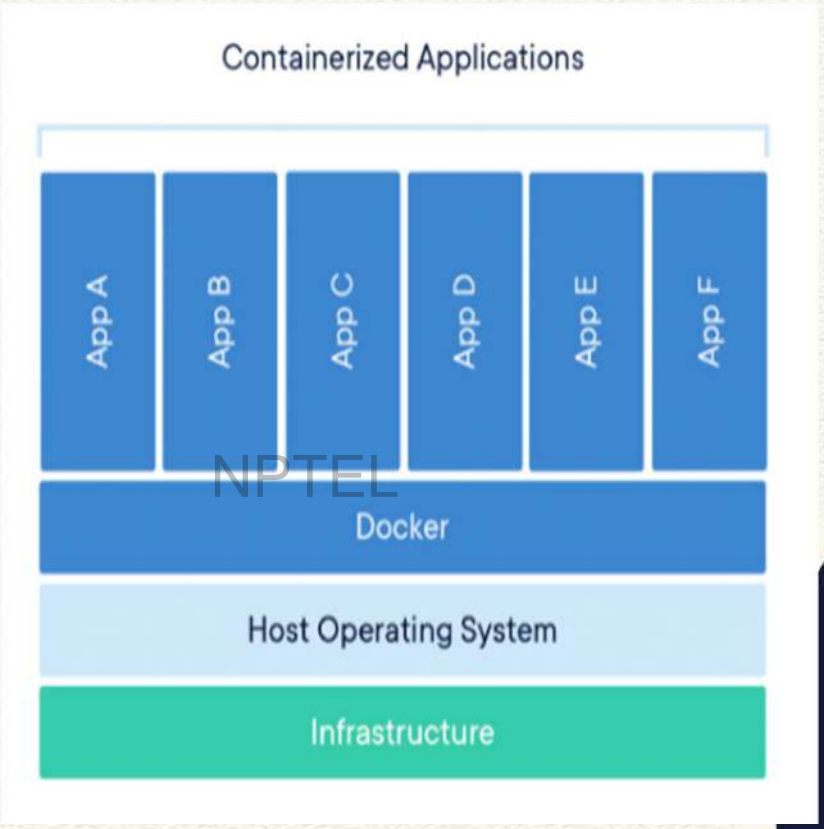
- VMs: a guest operating system such as Linux or Windows runs on top of a host operating system with access to the underlying hardware.
- Containers are often compared to virtual machines (VMs). Like virtual machines, containers allow one to package the application together with libraries and other dependencies, providing isolated environments for running your software services.
- However, the containers offer a far more lightweight unit for developers and IT Ops teams to work with, carrying a myriad of benefits.

- Containers are much more lightweight than VMs
- Containers virtualize at the OS level while VMs virtualize at the hardware level
- Containers share the OS kernel and use a fraction of the memory VMs require



Container

- A container is a sandboxed process that is isolated from all other processes on the host machine.
- A container is a runnable instance of an image.
- One can create, start, stop, move, or delete a container using the API (e.g. DockerAPI) or CLI.
- A container can be run on local machines, virtual machines or deployed to the cloud.

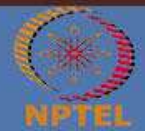


Ref: https://www.docker.com/resources/what-container

Kubernetes

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
- The name Kubernetes originates from Greek, meaning helmsman or pilot.
- Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions.
- However, Kubernetes is not monolithic, and these default solutions are optional and pluggable.
- Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

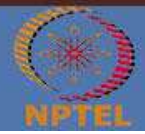
Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/



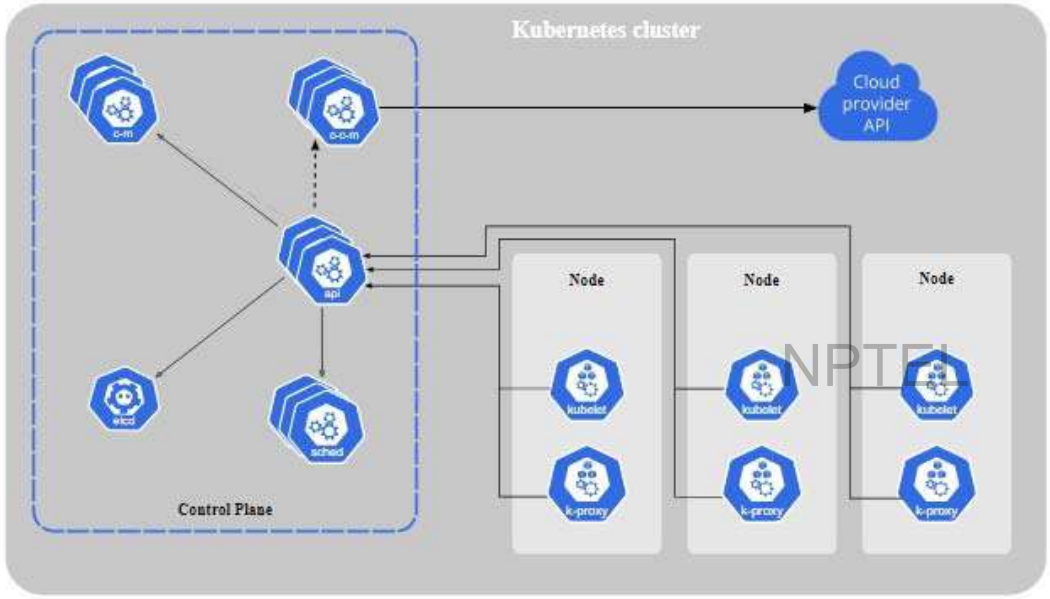
Kubernetes Components

- A Kubernetes cluster consists of a set of worker machines, called **nodes**, that run containerized applications. Every cluster has at least one worker node.
- The worker node(s) host the **Pods** that are the components of the application workload.
- The **control plane** manages the worker nodes and the Pods in the cluster.
- In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/



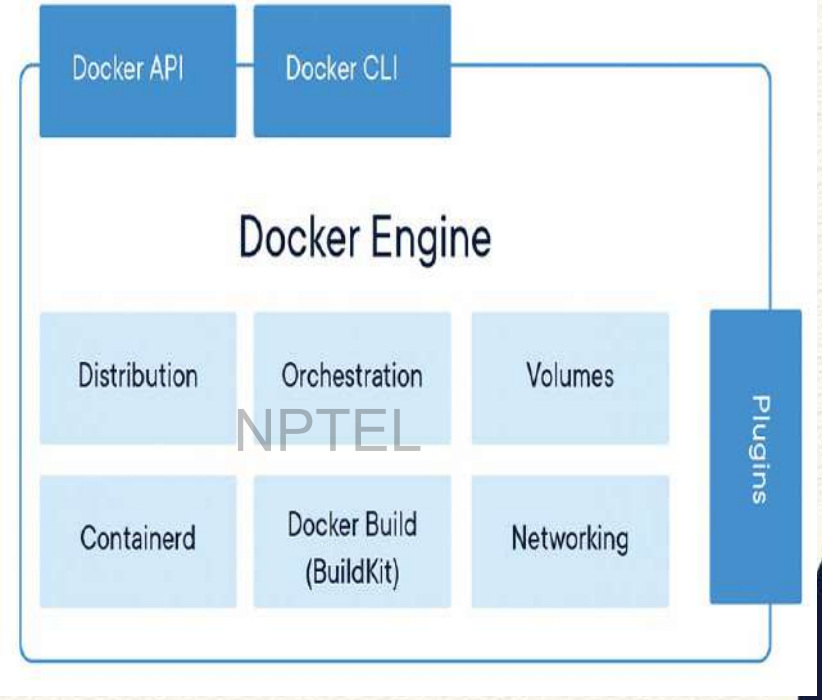
Kubernetes Cluster Components



Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Docker Engine

- Docker containers that run on Docker Engine:
- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

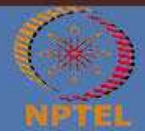


Ref: https://www.docker.com/resources/what-container

Dockers

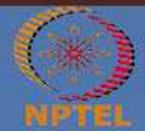
- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
- Container images become containers at runtime and in the case of Docker containers - images become containers when they run on **Docker Engine**.
- Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

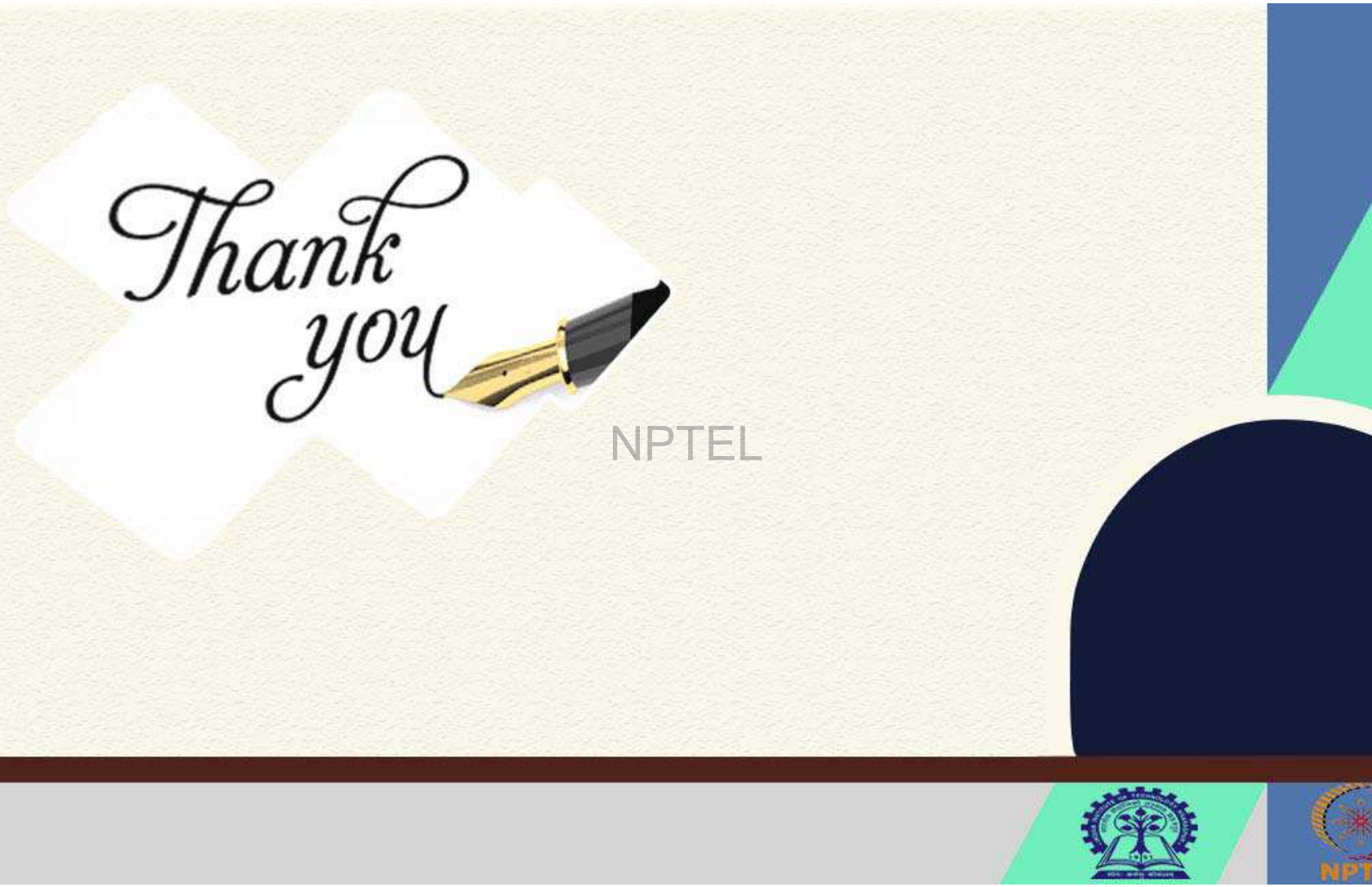
Ref: https://www.docker.com/resources/what-container



REFERENCES

- https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
- https://www.docker.com/resources/what-container
- https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler




Thank you
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Cloud Computing

Prof. Soumya K Ghosh

Department of Computer Science and Engineering

Module 10: Container

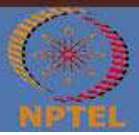
Lecture 50: Docker Container - Demo (Part-II)

CONCEPTS COVERED

- Docker Container - Demo

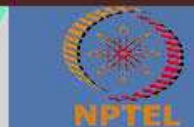
KEYWORDS

- Container
- Docker



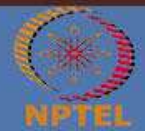
Docker Demo - II

NPTEL



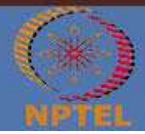
Introduction

- **Containers**
  - Standard unit of software
  - Packages up code and all its dependencies
  - Application runs quickly and reliably
- **Docker container image**
  - Lightweight
  - Standalone
  - Executable package of software
    - Includes everything needed to run an application



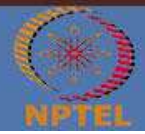
Demo - Objective

- MySQL and PHPMyAdmin on Docker platform
- MySQL
  - Widely used relational database package
  - Open-source
- PHPMyAdmin
  - A graphical user interface
  - Web-based
  - Connects to MySQL database
  - Widely used for managing MySQL databases



Standalone System (No Container)

- **Separate installation for**
  - MySQL
  - Web Server (Apache)
  - PHP
  - PHPMyAdmin
- **Transferring to other machine/ system**
  - Separate installation
  - Backup of data from old MySQL server
  - Restore the backup to new MySQL server

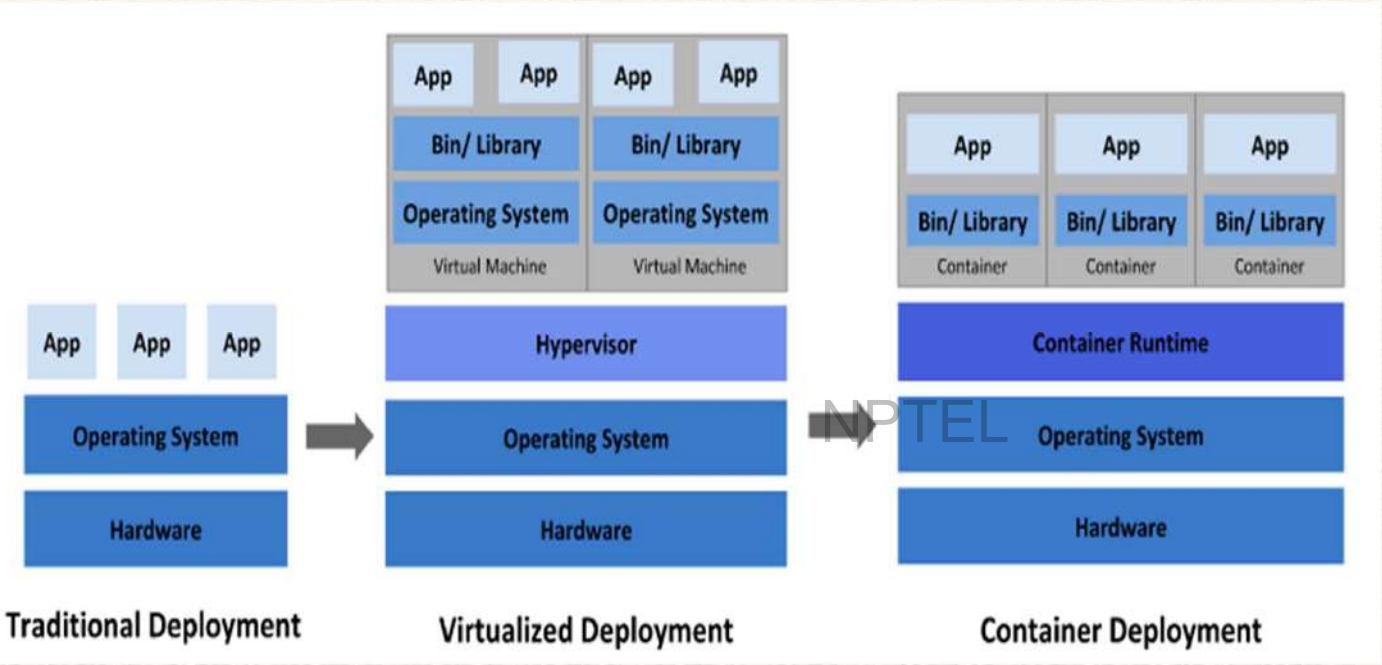


Containers – Major Benefits

- **Separation of responsibility**: Containerization provides a clear separation of responsibility, as developers focus on application logic and dependencies, while IT operations teams can focus on deployment and management instead of application details such as specific software versions and configurations.
- **Workload portability**: Containers can run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or on physical servers; on a developer's machine or in data centers on-premises; and of course, in the public cloud.
- **Application isolation**: Containers virtualize CPU, memory, storage, and network resources at the operating system level, providing developers with a view of the OS logically isolated from other applications.

Ref: https://cloud.google.com/learn/what-are-containers

Application Deployment



Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

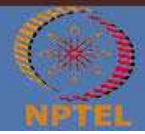
Traditional – Virtualized – Container Deployments

- **Traditional deployment :** Applications run on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.
- **Virtualized deployment :** Allows to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs. It allows better utilization of resources in a physical server and allows better scalability. Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.
- **Container deployment:** Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. A container has its own filesystem, share of CPU, memory, process space, and more. As containers are decoupled from the underlying infrastructure, they are portable across clouds and different OS distributions.

Containers and VMs

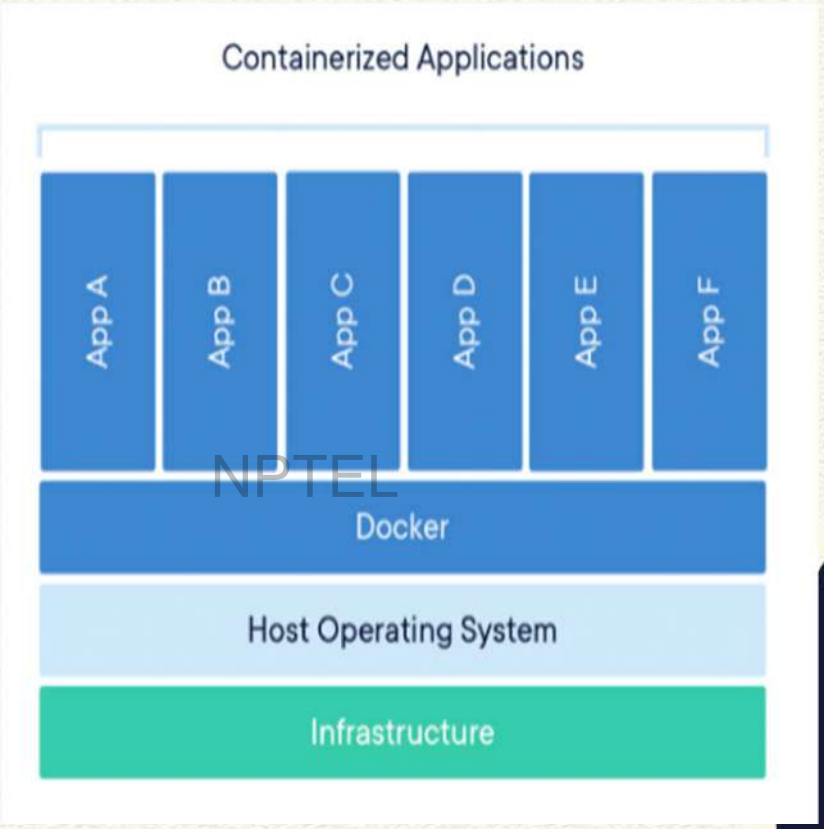
- VMs: a guest operating system such as Linux or Windows runs on top of a host operating system with access to the underlying hardware.
- Containers are often compared to virtual machines (VMs). Like virtual machines, containers allow one to package the application together with libraries and other dependencies, providing isolated environments for running your software services.
- However, the containers offer a far more lightweight unit for developers and IT Ops teams to work with, carrying a myriad of benefits.

- Containers are much more lightweight than VMs
- Containers virtualize at the OS level while VMs virtualize at the hardware level
- Containers share the OS kernel and use a fraction of the memory VMs require



Container

- A container is a sandboxed process that is isolated from all other processes on the host machine.
- A container is a runnable instance of an image.
- One can create, start, stop, move, or delete a container using the API (e.g. DockerAPI) or CLI.
- A container can be run on local machines, virtual machines or deployed to the cloud.

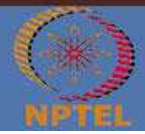


Ref: https://www.docker.com/resources/what-container

Kubernetes

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
- The name Kubernetes originates from Greek, meaning helmsman or pilot.
- Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions.
- However, Kubernetes is not monolithic, and these default solutions are optional and pluggable.
- Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

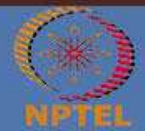
Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/



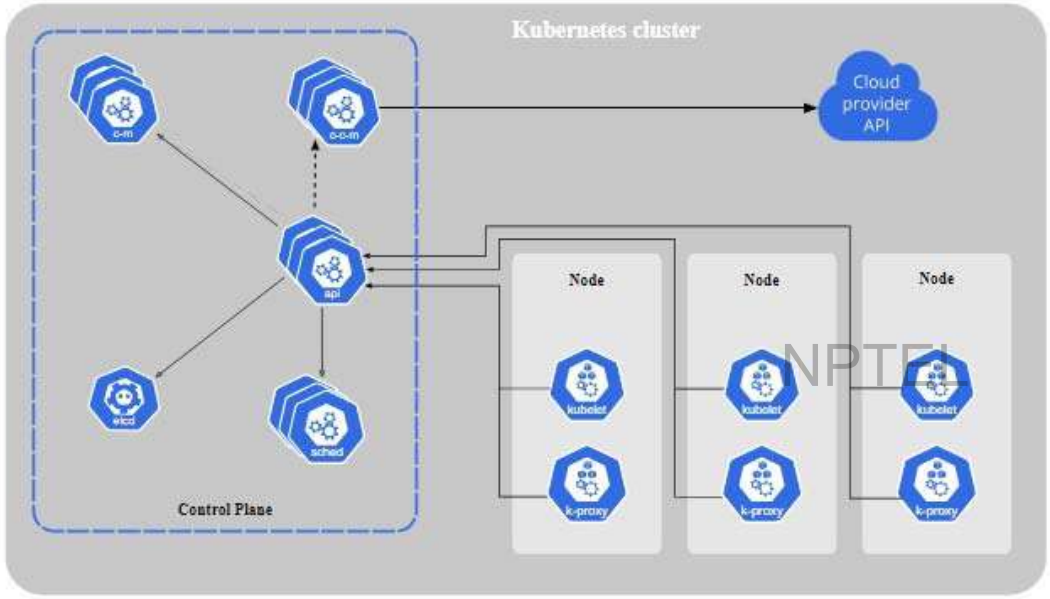
Kubernetes Components

- A Kubernetes cluster consists of a set of worker machines, called **nodes**, that run containerized applications. Every cluster has at least one worker node.
- The worker node(s) host the **Pods** that are the components of the application workload.
- The **control plane** manages the worker nodes and the Pods in the cluster.
- In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/



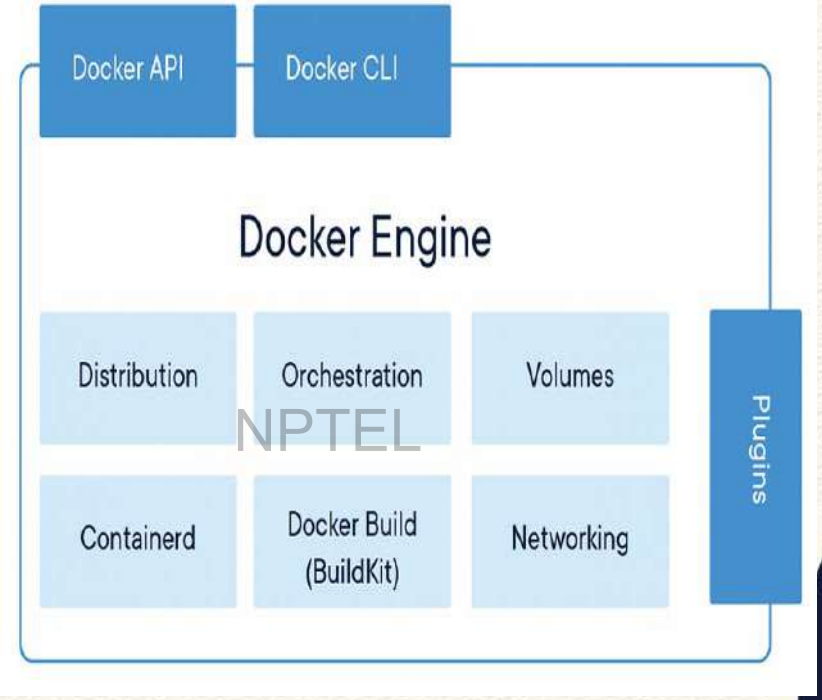
Kubernetes Cluster Components



Ref: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Docker Engine

- Docker containers that run on Docker Engine:
- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

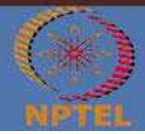


Ref: https://www.docker.com/resources/what-container

Dockers

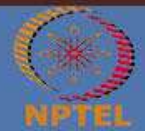
- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
- Container images become containers at runtime and in the case of Docker containers - images become containers when they run on **Docker Engine**.
- Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

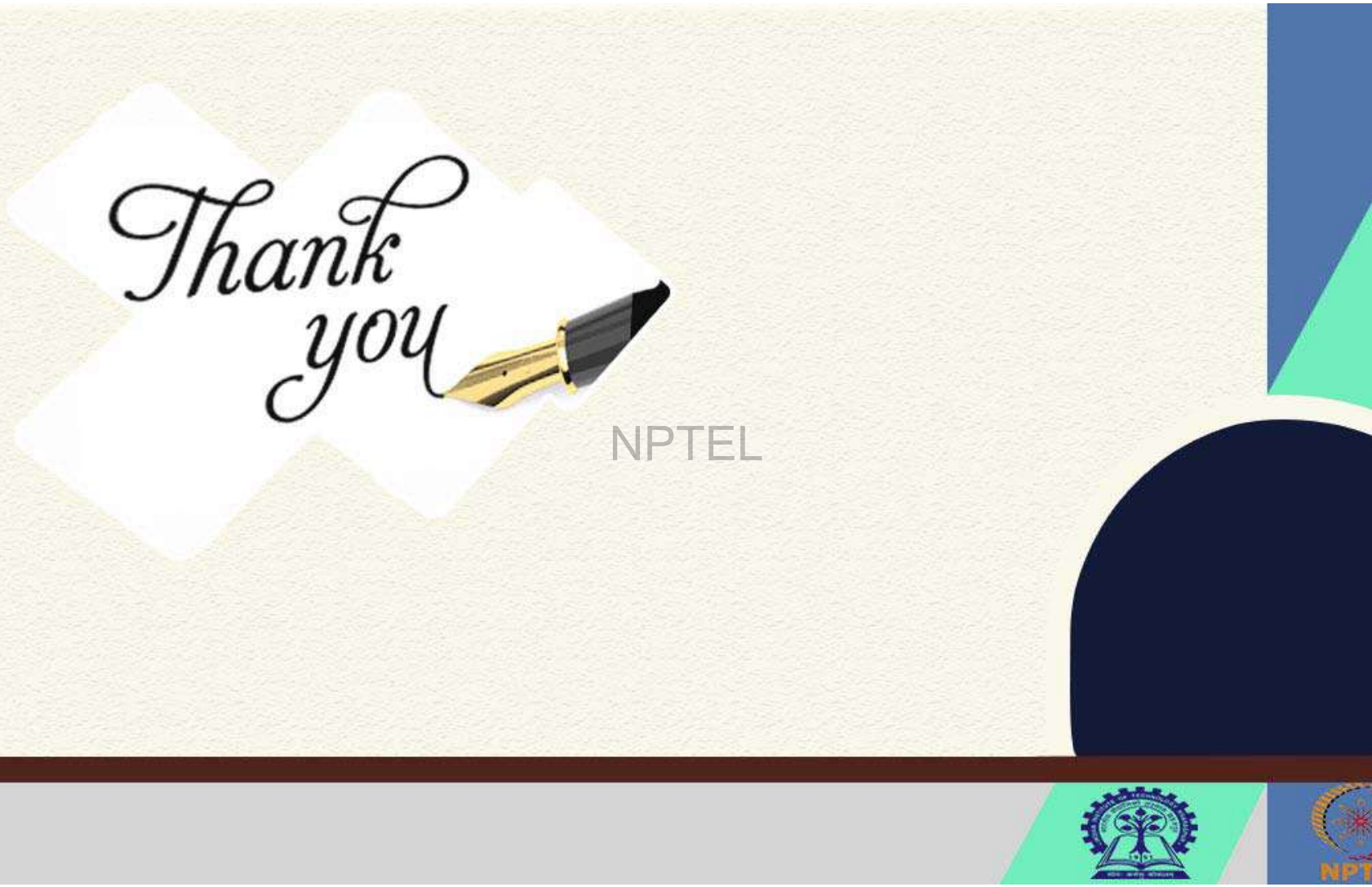
Ref: https://www.docker.com/resources/what-container



REFERENCES

- https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
- https://www.docker.com/resources/what-container
- https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler




Thank you
NPTEL