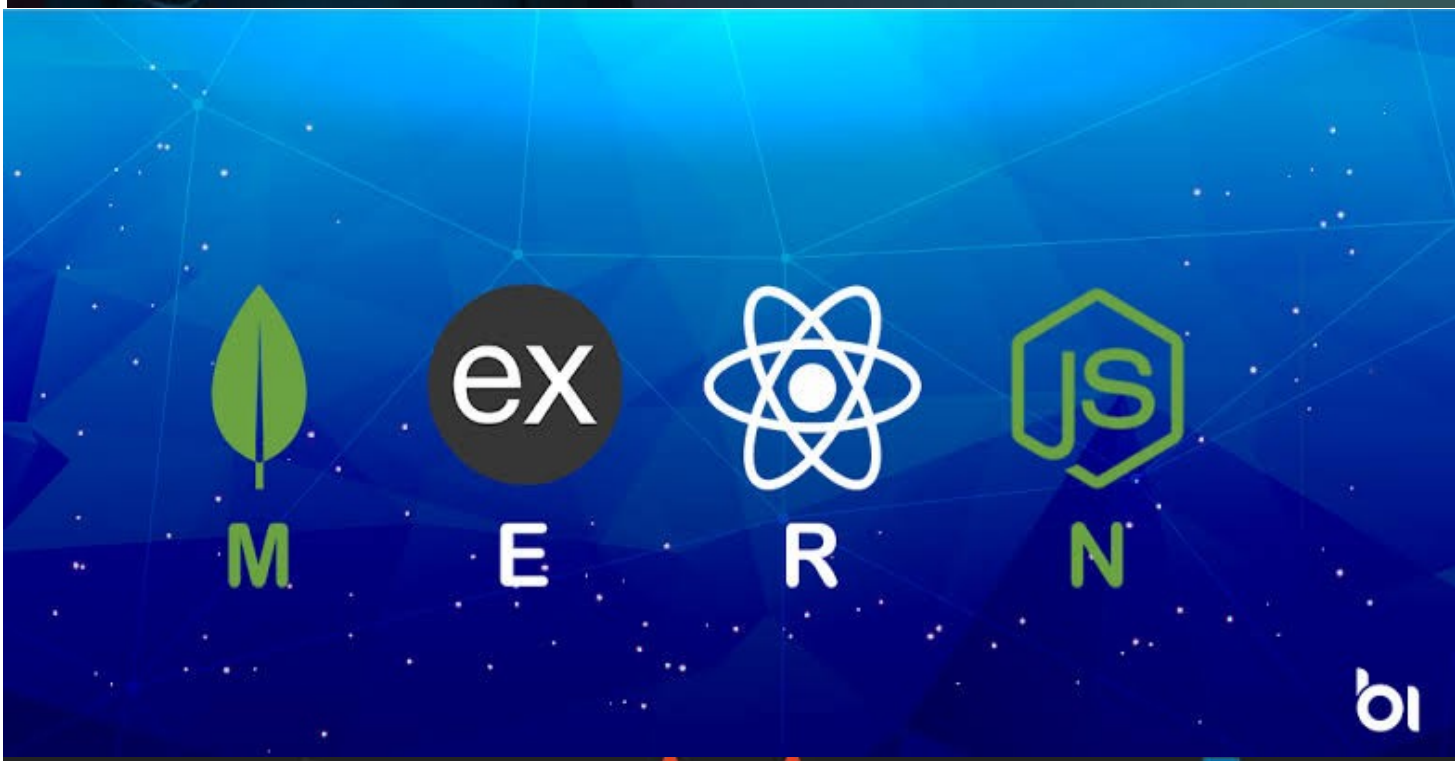
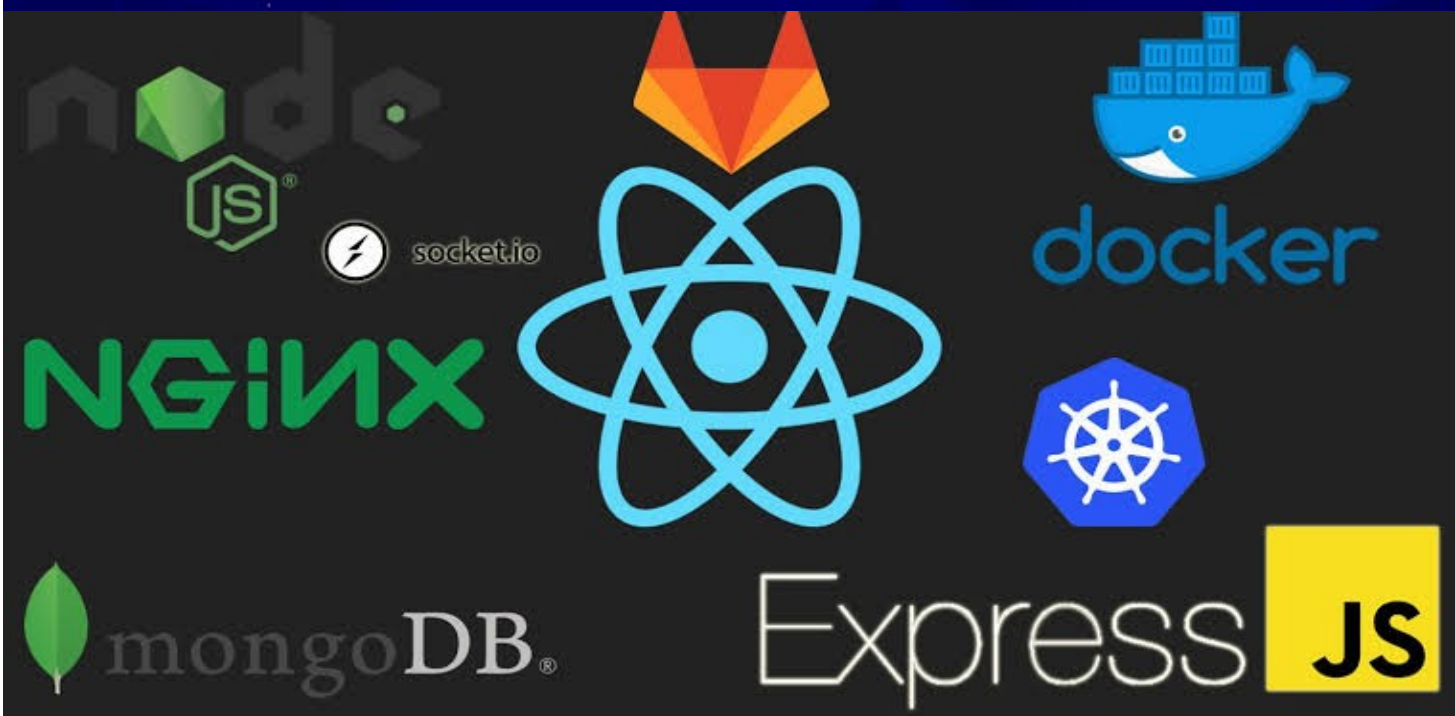

CLOUD NATIVE
COMPUTING FOUNDATION


ex
M
E
R
N


socket.io
docker
NGINX
mongoDB.
Express
JS

Course Contents

- Cloud Native Computing:
  - Linux
  - Docker
  - Kubernetes

- Front & Back End Development:
  - HTML5
  - CSS3
  - Javascript
  - Git

- MERN Stack Development:
  - MongoDB
  - Express.js
  - React.js
  - Node.js

- Other Technologies:
  - React Native
  - Firebase
  - Jamstack

CLOUD NATIVE COMPUTING

(The Modern Way To Develop Software)

TRADITIONAL WAY OF COMPUTING?

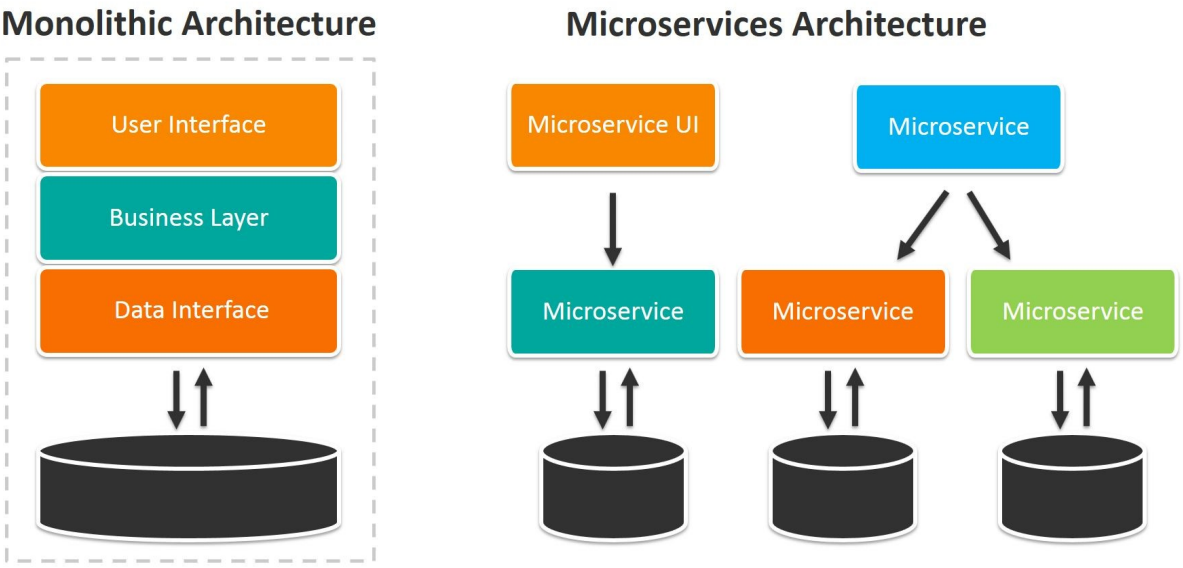
- Before we start discussing Cloud native computing, let’s first look at what was happening as a traditional way of computing.
- Traditional way/architecture of computing is also known as Monolithic computing.
- As a definition word monolithic means “very large, united, and difficult to change”.
- A monolithic architecture is the traditional unified model for the design of a software program.
- Monolithic, in this context, means composed all in one piece.
- Components/Layers of the program are interconnected and interdependent.
- In a tightly-coupled architecture, each component and its associated. components must be present in order for code to be executed or compiled.
- Application is too large and complex to fully understand and made changes fast and correctly.
- You must redeploy the entire application on each update.
- The size of the application can slow down the start-up time.
- Another problem with monolithic applications is reliability. Bug in any module can potentially bring down the entire process.
- Monolithic applications has a barrier to adopting new technologies. Since changes in frameworks or languages will affect an entire application it is extremely expensive in both time and cost.

MICROSERVICE ARCHITECTURE:

- The microservice architectural style is an approach to developing a single application as a suite of small services.
- Each runs in its own process and communicates with lightweight mechanisms, often an HTTP resource API.

- Microservices do have distinct advantages:

1. Better Organization:
    - Each microservice has a very specific job, and it is not concerned with the jobs of other components.
2. Decoupled:
    - Decoupled services are also easier to change, update and re-configure to serve the purposes of different apps.
    - They also allow for fast, independent delivery of individual parts within a larger, integrated system.
3. Performance:
    - Under the right circumstances, microservices can also have performance advantages depending on how they’re organized.
    - It’s possible to isolate hot services and scale them independently of the rest of the app.



WHAT IS CLOUD?

- In single line cloud can be described as a “communications network”.
- The word "cloud" often refers to the Internet, and more precisely to some datacenter full of servers that is connected to the Internet.

- A cloud can be a wide area network (WAN) like the public Internet or a private, national or global network. The term can also refer to a local area network (LAN) within an organization.
- Type of clouds

  1. Private Cloud
     - Deploying cloud computing internally,
     - Private cloud employs cloud computing within a company's own local or wide area networks.

  2. Public Cloud
     - A cloud computing service on the Internet that is available to the general public.
     - Commercial cloud providers like Amazon, Google cloud, Azure etc.

  3. Hybrid Cloud
     - The use of both private and public clouds to provide an organization's computing needs.

WHAT IS CLOUD NATIVE?

- Cloud Native Computing Foundation (CNCF) which is an open source software foundation dedicated to making cloud native computing universal and sustainable, describe cloud native as,

  “Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds”.
- Other definition of Cloud Native Computing could be:

  “An approach that builds software applications as microservices and runs them on a containerized and dynamically orchestrated platform to utilize the advantages of the cloud computing model”.
- Let’s get into detail and simplify to understand what we just heard.
- CLOUD NATIVE is about HOW applications are created and deployed, NOT WHERE.
- Cloud-native app development typically includes.

  ○ Devops,

  ○ Agile methodology,

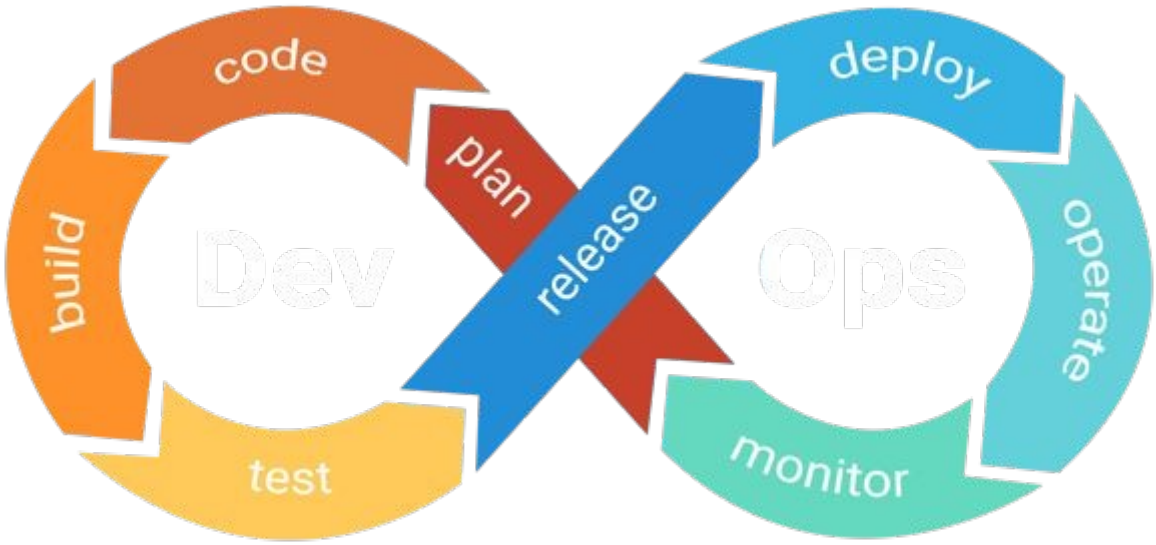
  ○ Microservices,

- Cloud computing platforms,
- Containerize application,
- Orchestration system,
- Continuous delivery

- In short, every new and modern method of application deployment.

DevOps:

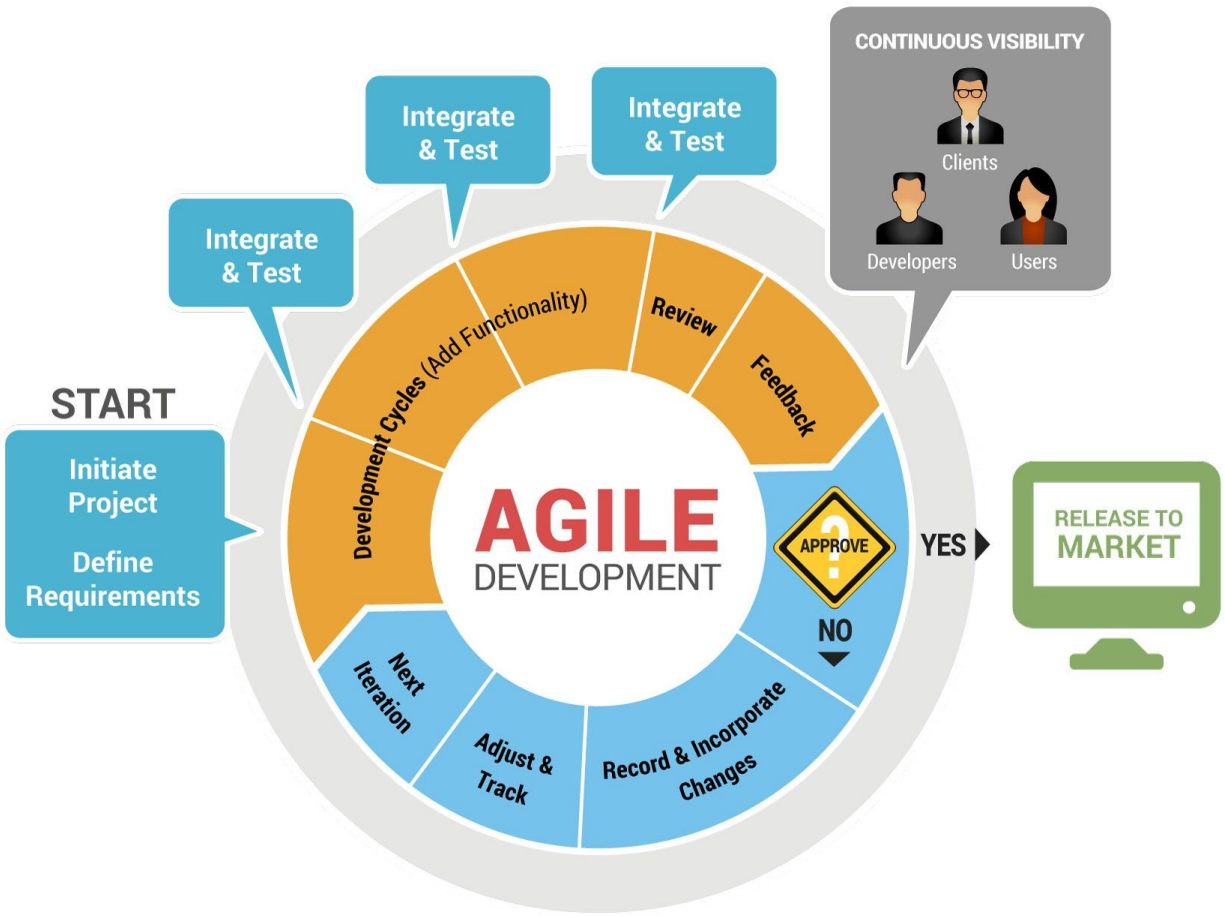
- Developer-cum-Operations



Agile Development:

- Agile methodology is described as an “iterative” and “incremental” approach.
- Agile developers visualize the software as a combination of complex parts that interacts with each other rather than a large block of structure.
- Actually, in waterfall method, development team will get only single chance to get each phase (like design, development, testing etc) of a project.
- Whereas in an agile methodology, these phases are continually revisited periodically to identify/understand the project’s progress and direction.

- The “inspect-and-adapt” approach from Agile methodology, greatly reduces development costs and time to market the product because here teams can develop the software while gathering changes in requirements.
- The stakeholders can provide feedback to the development team to improve the quality of the product.
- Agile development does save lot of resource which could have spent on something not needed anymore.



MICROSERVICES-ORIENTED:

- Cloud native applications are built as a system of microservices.
- The general idea of this architectural style is to implement a system of multiple, relatively small applications. These are called microservices.
- They work together to provide the overall functionality of your system.
- Each microservice realizes

○ Exactly one functionality,

○ Has a well-defined boundary and API (Application programming interface - used for communication),

○ Gets developed and operated by a relatively small team.

Benefits of Microservices:

- A lot easier to implement and understand a smaller application that provides one functionality, instead of building a large application that does everything.
- That speeds up development and makes it a lot easier to adapt the service to changed or new requirements.
- You need to worry a lot less about unexpected side effects of a seemingly small change, and you can focus on the development task at hand.
- It also allows you to scale more efficiently.
- And even if you only use a small part of the monolith, you still need to acquire additional resources for the other, unused parts but in a cloud environment, you pay for the usage of hardware resources.

CHALLENGES USING MICROSERVICES:

- There is a saying: There's no such thing as a Free Lunch.
- Microservices remove some complexity from the services themselves and provide better scalability, but you’re now building a distributed system.
- That adds a lot more complexity on the system level.
- To make sure that dependent services find each other and communicate efficiently its a challenging task when number of microservices are many.
- You also need to handle slow or unavailable services so that they don’t affect the complete system.
- The distributed nature of your system also makes it a lot harder to monitor and manage your system in production.
- Instead of a few monoliths, you now need to monitor a system of microservices,and for each service, there might be several instances that run in parallel.

CLOUD COMPUTING PLATFORMS:

- Cloud computing is the ON demand availability of computer system resources, especially data storage and computing power, without direct active management by the user.
- The term is generally used to describe data centers commercially available to many users over the Internet, they are Cloud Computing Platforms.
- Large clouds, predominant today, often have functions distributed over multiple locations from central servers. If the connection to the user is relatively close, it may be designated an edge server.
- An edge server also called content delivery network or content distribution network (CDN) is a geographically distributed network of proxy servers and their data centers. The goal is to provide high availability and high performance by distributing the service spatially relative to end-users.
- Clouds may be limited to.
    - A single organization (enterprise clouds).
    - Be available to many organizations (public cloud).
    - A combination of both (hybrid cloud).
- The largest public cloud is Amazon AWS. There are many others like Google cloud, microsoft Azure,Alibaba cloud, IBM cloud etc.
- Advocates of public and hybrid clouds note that cloud computing allows companies to avoid or minimize up-front IT infrastructure costs.
- Experts also claim that cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand.
- Cloud providers typically use a "pay-as-you-go" model.

CONTAINER:

- A Container is a runtime instance of an image.
- An Image is an executable package that includes everything needed to run an application.
    - the code,
    - a runtime,
    - Libraries,

    - environment variables,
    - and configuration files
- Making a container using image for any application is called Containerization.
- Docker is widely use to containerize your application.

CONTAINERIZATION:

- Containerization is increasingly popular because containers are:
    - Flexible: Even the most complex applications can be containerized.
    - Lightweight: Containers leverage and share the host kernel.
    - Interchangeable: You can deploy updates and upgrades on-the-fly.
    - Portable: You can build locally, deploy to the cloud, and run anywhere.
    - Scalable: You can increase and automatically distribute container replicas.
    - Stackable: You can stack services vertically and on-the-fly.

ORCHESTRATION:

- Deploying your application with all dependencies into a container is just the first step.
- Every ease comes with it's own challenges, making app containerized solves the deployment problems you had previously, but new challenges includes:
    - Scaling app based on the current load of your system isn't that easy. You need to
    - monitor your system,
    - trigger the startup or shutdown of a container,
    - make sure that all required configuration parameters are in place,
    - balance the load between the active application instances
    - share authentication secrets between your containers.

- Doing all of that manually requires a lot of effort and is too slow to react to unexpected changes in system load.
- You need to have the right tools in place that automatically do all of this.
- This is what the different orchestration solutions are built for.
- One of the most popular one is Kubernetes.

CONTINUOUS INTEGRATION /DELIVERY / DEPLOYMENT:

- CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development
- The main concepts attributed to CI/CD are
  - ○ CI is Continuous Integration,
  - ○ The “CD” refers to continuous delivery and/or continuous deployment
- CI/CD is a solution to the problems integrating new code can cause for development and operations teams.
- Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of apps, from integration and testing phases to delivery and deployment.
- Taken together, these connected practices are often referred to as a “CI/CD pipeline”.
- **Jenkins** is most popular tool used for CI/CD.
- **Continuous Integration (CI):** is an automation process for developers. Successful CI means new code changes to an app are regularly built, tested, and merged to a shared repository.
- **Continuous Delivery** usually means a developer’s changes to an application are automatically bug tested and uploaded to a repository (like GitHub or a container registry), where they can then be deployed to a live production environment.
- **Continuous Deployment** (the other possible “CD”) can refer to automatically releasing a developer’s changes from the repository to production, where it is usable by customers.

Linux

Linux

Book: Easy Linux for Beginners By Felix Alvaro

WHAT IS AN OPERATING SYSTEM?

- A computer, complete with all its parts – the CPU, mouse, monitor, and keyboard – will not work without a central program that will piece it all together.
- In order to use a PC, you need a software inside which will take care of making the hardware work for you.
- A special kind of software which is between the hardware of the PC, and the programs that you want to use and work with.
- This piece of software is the Operating System, or more easily referred to as just an OS.
- In short, an operating system is the software that brings together a computer's hardware and the different programs that you want to install on it.
- Without it, when you booted up your PC, you would not get anything on the monitor, and neither mouse nor keyboard will work.

WHAT OPERATING SYSTEM DOES?

- Initiate user interfaces.

  ○ An OS offers users ways to access the system either via a command line or a graphical user interface (GUI) Establish file systems.

  ○ The OS handles the management of files (access, directories, and structure), including the access to the file system.

- Manage access and user authentication.

  ○ An OS allows for creating user accounts with different permissions for access to files and processes.

- Provide a platform for administrative use.
  - A computer's OS provides a platform for the administrator to
    - Add users,
    - Allocate disk space,
    - Install software,
    - Perform activities to manage the computer
- Start-up services

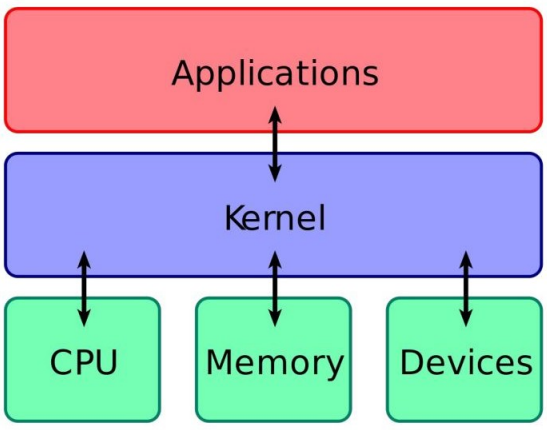
  ○ The OS manages several processes running in the background known as daemon processes.

WHAT IS LINUX?

- Linux is an operating system, like the examples mentioned in the previous slide, and is often described as Unix-like.
- The difference between Linux and other operating systems lies in the fact that Linux is an open-source operating system.
- This means that Linux is continuously developed collaboratively.
- Unlike Windows and MacOS which are both tied to the respective companies (Windows and Apple).
- Not one single company owns Linux' development and support.
- Different companies sharing research, development, and the associated costs to upgrade linux operating system.
- This open source cooperation among companies and developers has led to making.
- Linux one of the best ecosystems for use from small digital wristwatches to servers and supercomputers.
- Based on statistics, there are at least 100 companies and more than 1000 developers who work together for every kernel release.
- Linux is composed of a kernel, the core control software, plus plenty of libraries and utilities that provide different features.
- The kernel is the lowest level of the operating system
- The kernel is the main part of the operating system and is responsible for translating the command into something that can be understood by the computer.

The main functions of the kernel are:

- Memory management
- Network management
- Device driver
- File management
- Process management



- Linux is available through many distributions. These are what we can call Linux flavours.
- Distributions are groups of specific kernels and programs. The most popular ones include Arch, SUSE, Ubuntu, and Red Hat.

- Even Android a mobile operating system developed by Google is based on a modified version of the Linux kernel along with other open source software.

ADDITIONAL CHARACTERISTICS:

In addition to the tasks performed by an operating system, Linux has the following characteristics:

- Supports clustering
  - ○ Multiple Linux systems can be configured to appear as one system from the outside.
  - ○ Service can be configured among clusters and still offer a seamless user experience.
- Runs virtualization
  - ○ Virtualization allows one computer to appear as several computers to users.
  - ○ Linux can be configured as a virtualization host.
    - ■ Where you could run other OS such as Windows, Mac OS, or other Linux systems.
    - ■ All the virtualized systems appear as separate systems to the outside world.
- Cloud Computing
  - ○ Linux can handle complex, large-scale virtualization needs including.
    - ■ Virtual networking: a technology that facilitates data communication between two or more virtual machines (VM),
    - ■ Networked storage,
    - ■ Virtual guests (supports guest operating system using VM)
- Options for Storage
  - ○ Data need not always be stored in your computer’s hard disk.
  - ○ Linux offers different local and networked storage options.

Through collaborative works, Linux is now one of the most powerful operating systems. Data shows that 98.8% of the world’s fastest systems use the Linux kernel.

LINUX VS OTHER OS:

- Cost
  - ○ Other than Linux commercial distributions all other linux flavours are free.
- Viruses
  - ○ Linux hardly gets any viruses.
  - ○ With many developers working on Linux, there are more eyes focused on seeing security flaws.
- System Stability
  - ○ Linux is used in servers and supercomputers which cannot afford server restarts.
  - ○ Large-scale systems can go on for years without restarting the serv.er
- Installation
  - ○ Linux flavours comes with text editor, spreadsheet, presentation program, photo editor, web browser, movie player, PDF reader, and the like.
  - ○ In other OS like windows etc we have to install all the other software that you need one-by-one.
- Support
  - ○ Linux has a large community online where new users can get information, read FAQs, and ask questions if there are programs or features that you think are not working right.

LINUX ARCHITECTURE:

Linux architecture can be divided into two spaces

- ○ The User Space,
- ○ the Kernel Space

- User Space
  - ○ This is where the applications are used.
  - ○ The GNU C library, in the User space, is the interface that connects to the kernel and transitions between User and Kernel space.
  - ○ This uses all the available memory.

- Kernel Space

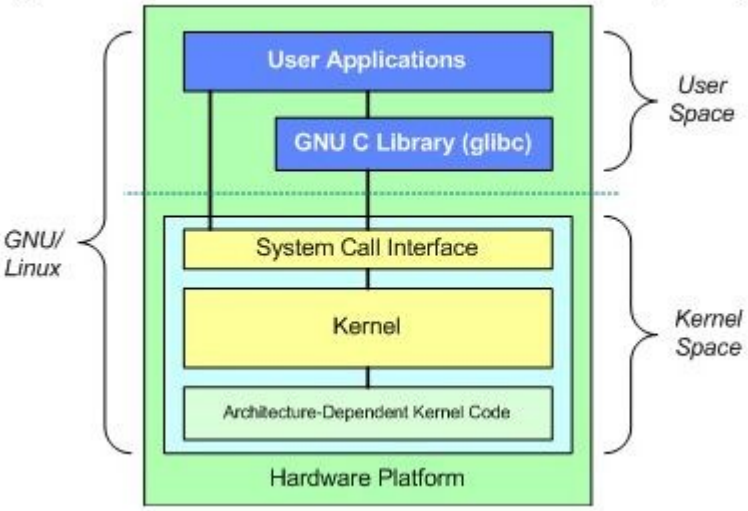
  All Kernel services are processed here. The Kernel space is further divided into 3.

  - System Call Interface
    - A User process can access Kernel space through a System Call.
    - When a System Call is performed, arguments are passed from User to Kernel space. This is the layer that implements basic functions.

  - Kernel Code
    - This is the architecture-independent code, and can be seen in all architectures that Linux supports.

  - Architecture-Dependent Kernel Code
    - This is the layer for platform-specific codes.

Figure 2. The fundamental architecture of the GNU/Linux operating system



LINUX SHELL:

- Linux distribution uses the root username for administrator access.
- The desktop that comes up is either KDE or GNOME.
- The GUI for KDE or GNOME provides the best way to explore Linux through icons, windows, and pointers.
- Alternate to it is Terminal / Console window.
- This is called shell.
- The shell is where you can run executable files and shell scripts.
- The shell is also what we call the command line. Commands are written using the general syntax below:
- command option1 option2 . . . optionN
- For example you can try following command

```
imtiaz@computer-scientist:~$ uptime
 23:23:33 up  8:31,  1 user,  load average: 1.37, 1.24, 1.20
```

- **uptime** is the command that shows the duration that the computer has been up. In this case.

- For example you can try following command

```
imtiaz@computer-scientist:~$ uname -srv
Linux 5.3.0-28-generic #30~18.04.1-Ubuntu SMP Fri Jan 17 06:14:09 UTC 2020
```

- **uname** is the command to show the operating system name.
  - -s (print the operating system name),
  - -r (print the operating system release),
  - -v (print the operating system version) are options that you can use for the uname command.

```
imtiaz@computer-scientist:~$ uname --help
Usage: uname [OPTION]...
Print certain system information.  With no OPTION, same as -s.

  -a, --all                print all information, in the following order,
                             except omit -p and -i if unknown:
  -s, --kernel-name        print the kernel name
  -n, --nodename           print the network node hostname
  -r, --kernel-release     print the kernel release
  -v, --kernel-version     print the kernel version
  -m, --machine            print the machine hardware name
  -p, --processor          print the processor type (non-portable)
  -i, --hardware-platform  print the hardware platform (non-portable)
  -o, --operating-system   print the operating system
```

- To know about the options that you can use for a particular command, you can use the man command.

imtiaz@computer-scientist:~$ man hostname

UNAME(1)

User Commands

UNAME(1)

NAME

uname - print system information

SYNOPSIS

uname [OPTION]...

DESCRIPTION

Print certain system information. With no OPTION, same as -s.

-a, --all

print all information, in the following order, except omit -p

and -i if unknown:

-s, --kernel-name

print the kernel name

-n, --nodename

ROOT USER:

- In computing, the superuser is a special user account used for system administration.
- The root user can do many things an ordinary user cannot for example install new software, removing any existing application, making change in any system settings etc.
- Depending on the operating system (OS), the actual name of this account might be root, administrator, admin or supervisor.
- To switch to root while in the shell, enter the command su - and then input your root password.
- Changing to root password while in the shell environment will allow you to run tasks that only administrators and superusers can do.

imtiaz@computer-scientist:~$ su

Password:

root@computer-scientist:/home/imtiaz# exit

exit

imtiaz@computer-scientist:~$

imtiaz@computer-scientist:~$clear

LINUX FILESYSTEM:

- Linux organizes files using a hierarchical system.
- Files are stored in directories and these directories can also contain other directories.
- When you compare the Linux filesystem to Windows, you will find that there are no drive letters in Linux.
- All files are stored in a single root directory noted as "/" regardless of where the data is physically stored (hard drive, external drive, or CR-ROM).
- To find a file in Linux, you also need the information about the directory hierarchy known as the path name.

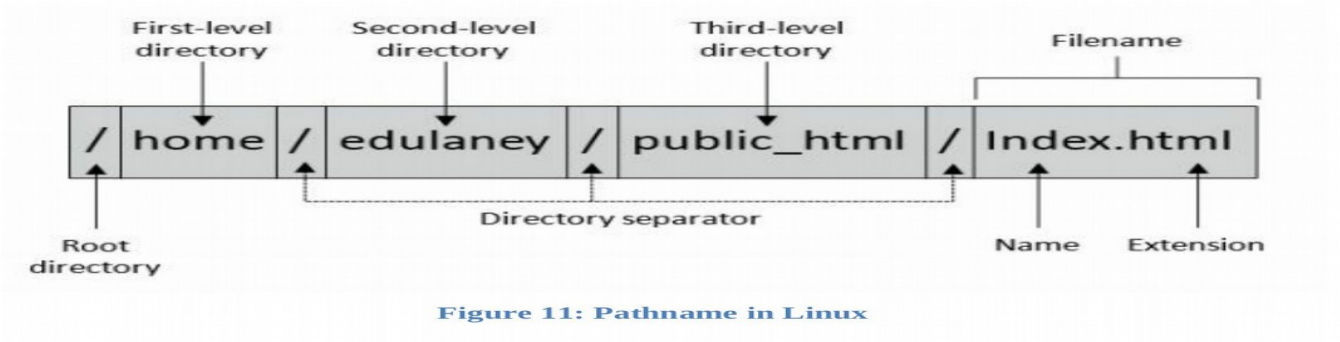


Figure 11: Pathname in Linux

Linux Shell Shortcut Keys:

- Ctrl + A

  Transfer Cursor to the beginning of the line.

- Ctrl + E

  Transfer Cursor to the end of the line.

- Delete

  The Character Under the Cursor is deleted.

- Ctrl + R

  Press Ctrl + R. Then use up/down keys to see previous used command.

- Tab

  Type unfinished command and press tab , complete command will automatically written.

- Double Tab

  Type one or more character and press double tab, it will display all commands starts with the given letter.

➢ While installation and for some commands permission will be denied, then use **sudo** before that command which give it permission after getting your password.

Linux Commands:

Sr. #	Command	Example
	Purpose	

Help Commands

1.	info <command-name>	info uptime
	Read online documentation of a command.	

2.	man <command-name>	man uptime
	Display manual of a command.	

3.	whatis <command-name>	whatis uname
	Show a short description of a commad.	

4.	alias <shortcut-name> = <command>	alias switchuser = su
	Assign a shortcut name of a command (till restart).	

5.	unalias <shortcut-name>	unalias switchuser
	Removes assigment .	

CPU Information Commands

6.	uname -a
	Display info about the machine, processor, OS.

7.	lscpu
	Also Display CPU info.

8.	cat /*proc*/cpuinfo
	Also Display CPU info.

9.	top
	Display CPU load and used memory averages.

Installation Commands

10.	apt-get install <package-name>	apt-get install vlc
	Download the package and install it.	

11.	apt-cache search <keyword> |grep GNOME	apt-cache search netbeans |grep GNOME
	Search a package related to your keyword.	

12.	apt-get update
	Install updates.

13.	apt-get upgrade
	For uprade.

File System Commands

14.	pwd
	Display current working directory

15.	ls
	Display all files and directories present on current directory.

16.	ls -a
	Display all files and directories present on current directory (including hidden).

17.	ls <path>	
	Display all files and directories present on given path.	

18.	ls -a <path>	
	Display all files and directories present on given path (including hidden).	

19.	cd ..
	Change working directory (from present to one directory back).

20.	cd ../..
	Go to two back level directory

21.	cd ~
	Go to home directory.

22.	cd Desktop
	Go to Desktop Directory

23.	cd <path>	cd /*home*/imtiaz/java
	Go to given directory. (If desired dir is not present in current working dir , then go to root dir (/) or use cd .. or cd <path> according to directories and move onwards i.e cd /*home/*imtiaz/java ,), (if desired dir is present in current working dir then write name of dir i.e cd java)	

Managing Files

24.	touch <file-name>	touch abc.txt
	Creates a file in current working directory.	

25.	touch .<file-name>	touch .abc.txt
	Creates a hidden file in current working directory.	

26.	touch <path><file-name>	touch /home/imtiaz/java/class.txt
	Creates a file to given directory.	

27.	touch <path><.file-name>	touch /home/imtiaz/java/.class.txt
	Creates a hidden file to given directory.	

28.	rm <file-name>	rm abc.txt
	Remove file from current working directory.	

29.	rm <path><file-name>	rm /home/imtiaz/java/program.txt
	Remove file from current working directory.	

30.	echo <text> > <file-name>	echo Hello World > abc.txt
	Write given text to the file which is present in current working directory. (if given file is not already present echo command creates a file and write text to file).	

31.	echo <text> >> <file-name>	echo Its My Program >> abc.txt
	Append text to the file which is present in current working directory.	

32.	echo <text> > <path><file-name>	echo Hello World > /home/imtiaz/program.txt
	Write given text to the file which is present in given directory.	

33.	echo <text> >> <path><file-name>	echo Hello >> /home/doc/abc.txt
	Append given text to the file which is present in given directory.	

34.	cat > <file-name>	cat > mydoc.txt
	Cat command is used to write text in file. Write cat then > sign and file name.then write text at the end press ctrl + c. if file is not present it will create and write text to the file.	

35.	cat > <path><filename>	cat > /home/doc/abc.txt
	Same as upper command but it do all thing to the given directory.	

36.	cat <file-name>	cat mydoc.txt
	Display contents of file present in current working directory.	

37.	cat <path><filename>	cat /home/doc/abc.txt
	Same as upper command but it do all thing to the given directory.	

38.	cat <file-name> | grep <text>	cat abc.txt | grep Hello
	Display contents of file , highlight given text (case sensitive)	

39.	cat <path><file-name> | grep <text>	cat doc/abc.txt | grep Hello
	Display contents of file which is present in given directory, highlight given text (case sensitive)	

40.	cat <file-name> | grep -i <text>	cat abc.txt | grep -i Hello
	Display contents of file , highlight given text (ignore case sensitive)	

41.	cat <path><file-name> | grep -i <text>	cat doc/abc.txt | grep -i Hello
	Display contents of file which is present in given directory, highlight given text (ignore case sensitive)	

42.	cp <file-name> <path>	cp xy.mp3 mymusic/new/
	Copy file to given directory.	

43.	cp <source><file-name> <destination-path>	cp doc/abc.txt info/
	Copy file from source path to destination.	

44.	cp <source-path><file-name> <destinaetion-path><new-name>	cp doc/abc.txt info/xyz.txt
	Copy file from source path to destination with given name.	

45.	mv <file-name> <path>	mv abc.txt info/
	Move file to given directory.	

46.	mv <source><file-name> <destination-path>	mv doc/abc.txt info/
	Move file from source path to destination.	

47.	mv <file-name> <new-name>	mv abc.txt xyz.txt
	Rename file.	

48.	mv <source-path><file-name> <new-name>	mv doc/abc.txt xyz.txt
	Rename file which is present in given path.	

49.	mv <source-path><file-name> <destination-path><new-name>	mv doc/abc.txt info/xyz.txt
	Move file from source path to destination with given name.	

50.	We can operation as creation, copy, move, remove on multiple files.
	i.e rm abc.txt xyz.txt or touch doc.txt doc1.txt

Managing Directories

51.	mkdir <dir-name>	mkdir mydoc
	Makes directory.	

52.	mkdir <path><dir-name>	mkdir /*home*/imtiaz/java/mydoc
	Makes dir present on specified path.	

53.	rmdir <dir-name>	rmdir mydoc
	Removes empty dir.	

54.	rm <path><dir-name>	rmdir /*home*/imtiaz/java/mydoc
	Removes empty dir present on specified path	

55.	mkdir -p <directories>	mkdir -p docs/linuxdoc/mydoc
	Makes multiple directories.	

56.	mkdir -p <path> <directories>	mkdir -p */home/imtiaz/cnc/linuxdoc*
	Makes multiple subdirectories.	

57.	rm -r <dir>	rm -r cnc
	Removes dir whether its empty or contains sub dir or files.	

58.	rm -r <path><dir>	rm -r */home/imtiaz/cnc*
	Removes dir present on given path whether its empty or contains sub dir or files.	

59.	cp -r <dir> <path>	cp -r cnc java/doc/
	Copy dir whether its empty or contains sub dir or files.	

60.	cp -r <path><dir> <path>	cp -r */home/imtiaz/cnc java/doc/*
	Copy dir present on given path whether its empty or contains sub dir or files.	

61.	mv -r <dir> <path>	cp -r cnc java/doc/
	Moves dir whether its empty or contains sub dir or files.	

62.	mv -r <path><dir> <path>	cp -r */home/imtiaz/cnc java/doc/*
	Moves dir present on given path whether its empty or contains sub dir or files.	

63.	We can use . (dot) if want to do anything on current working dir.
	i.e if we want to copy a file from any dir to our current working dir then we have to use . (dot) as our destination path.

64.	df -h
	Validate the disk space, -h make it human readable.

65.	du -sh
	Display current directory size.

66.	du -sh <dir-name>	du -h mydoc
	Display given directory size.	

67.	du -sh <path><dir-name>	du -h /home/imtiaz/java
	Display size of given dir which is present on specified dir.	

68.	du -h <file-name>	du -sh Test1.pdf
	Display size of given file.	

69.	du -h <path><filename>	du -sh /home/imtiaz/Desktop/doc.pdf
	Display size of given file which is present on specified dir.	

70.	du -h
	Display size of all files present on current working dir.

71	du -h <dir-name>	du -h Desktop
	Display size of all files present in specified dir (which is present on PWD (Present Working Directory))	

72.	du -h <path><dir-name>	du -h /Desktop/mydoc
	Display size of all files present in specified dir.	

Finding Files/Commands

73.	find <path> -name <file-name>	find /home/imtiaz/ -name mydoc.pdf
	Finds file or dir on specified path. if you want to search on PWD just place . (dot) as mentioned before.	

74.	whereis <exe-file>	whereis uname
	Search for executeable files.	

Commands For Managing Processes

75.	reboot
	For system reboot.

76.	shutdown
	For system shutdown.

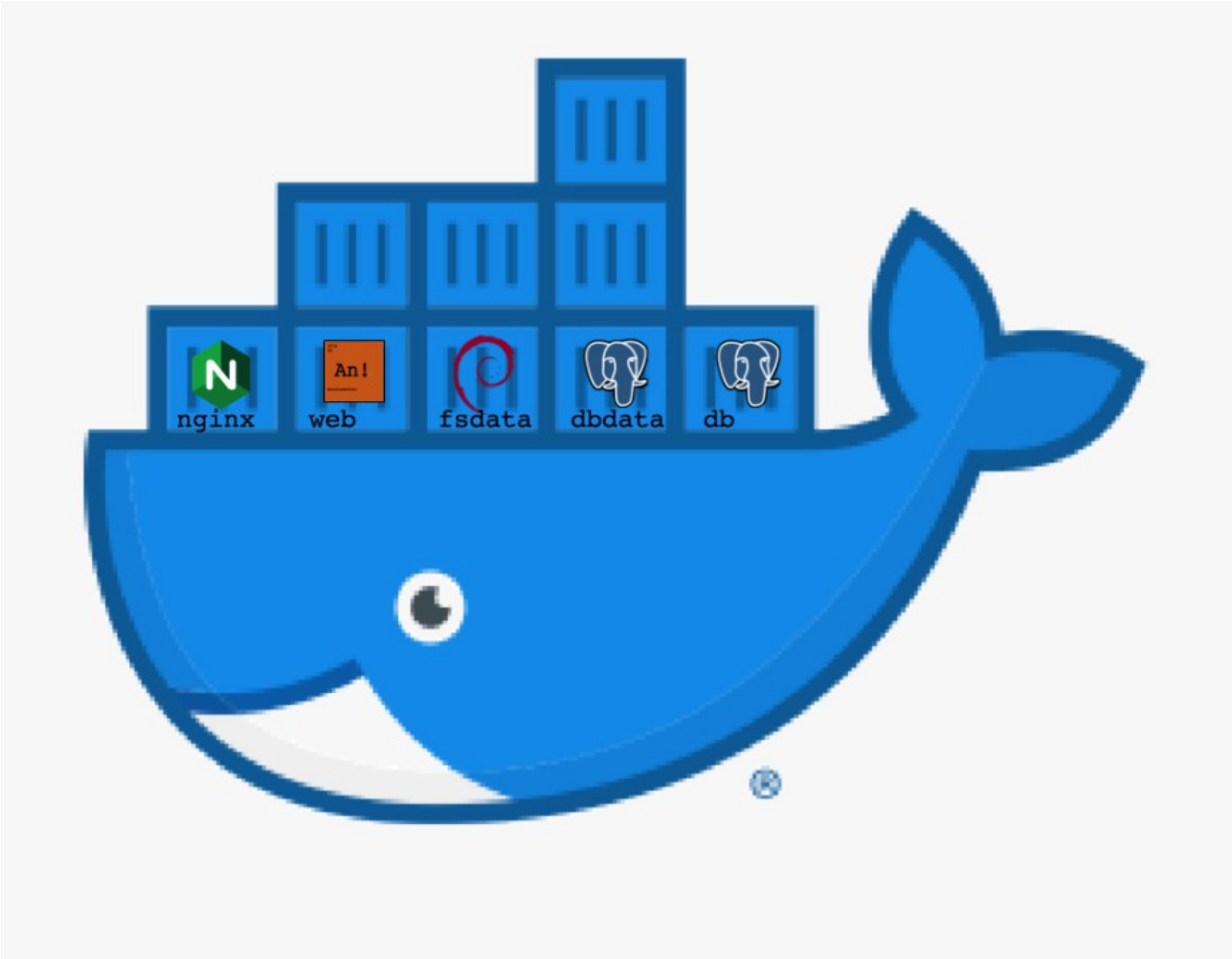
77.	free
	To check free memory.

78.	ps -aux | grep <package-name>	ps -aux | grep vlc
	To get PSID of a package.	

79.	kill -9 <PSID>	kill -9 4321
	To close the process forcefully.	

80.	ps <PSID>	ps 3932
	Display the name of package having specified PSID.	

Docker



Book: Docker Deep Dive by Nigel Poulton

The Bad Old Days:

- Applications run businesses. If applications break, businesses suffer and sometimes go away. These statements get truer every day!
- Most applications run on servers. And in the past, we could only run one application per server. The open-systems world of Windows and Linux just didn't have the technologies to safely and securely run multiple applications on the same server.The Bad Old Days
- So, the story usually went something like this...
- Every time the business needed a new application, IT would go out and buy a new server.
- And most of the time nobody knew the performance requirements of the new application!
- This meant IT had to make guesses when choosing the model and size of servers to buy.
- As a result, IT did the only thing it could do - it bought big fast servers with lots of resiliency.
- After all, the last thing anyone wanted - including the business – was under-powered servers. Under-powered servers might be unable to execute transactions, which might result in lost customers and lost revenue. So, IT usually bought bigger servers than were actually needed. This resulted in huge numbers of servers operating as low as 5-10% of their potential capacity. A tragic waste of company capital and resources!

  - Disadvantages

    1.Very costly.

    2.Resource wastage.

    3.Many servers to manage.

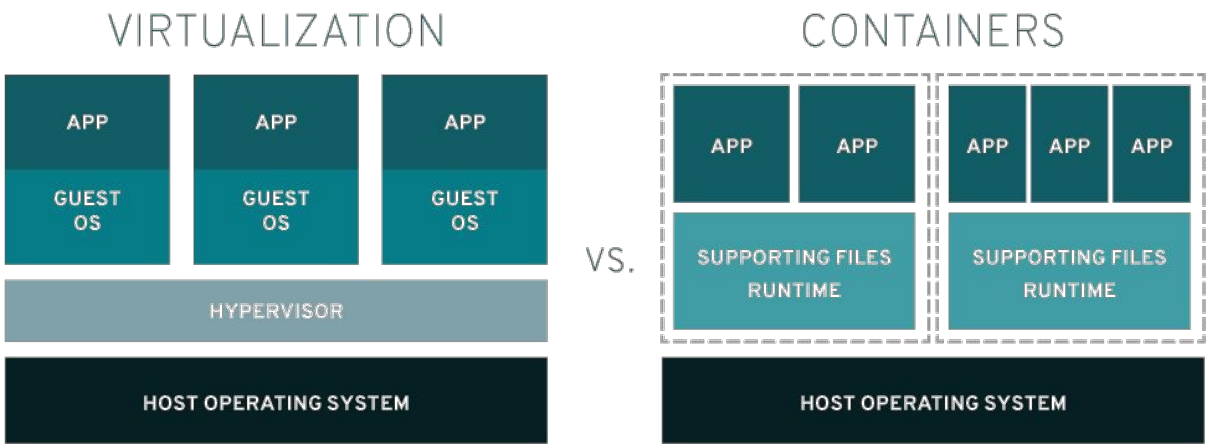
Hello Vmware!

- Amid all of this, VMware, Inc. gave the world a gift - the virtual machine (VM).
- And almost overnight the world changed into a much better place! We finally had a technology that would let us safely and securely run multiple business applications on a single server.
- This was a game changer! IT no longer needed to procure a brand new oversized server every time the business asked for a new application.

- More often than not they could run new apps on existing servers that were sitting around with spare capacity.
- All of a sudden, we could squeeze massive amounts of value out of existing corporate assets, such as servers, resulting in a lot more bang for the company's buck.

Vmwarts:

- But... and there's always a but! As great as VMs are, they're not perfect! The fact that every VM requires its own dedicated OS is a major flaw.
  - Every OS consumes CPU, RAM and Storage that could otherwise be used to power more applications.
  - Every OS needs patching and monitoring.
  - And in some cases, every OS requires a license.
- The VM model has other challenges too. VMs are slow to boot and portability isn't great - migrating and moving VM workloads between hypervisors and cloud platforms is harder than it needs to be.



Hello Containers:

- For a long time, the big web-scale players like Google have been using container technologies to address these shortcomings of the VM model.
- In the container model the container is roughly analogous to the VM.
- The major difference through, is that every container does not require a full-blown OS.

- In fact, all containers on a single host share the same OS Kernel. This frees up huge amounts of system resources such as CPU, RAM, and storage.
- It also reduces potential licensing costs and reduces the overhead of OS patching and other maintenance.
- Containers are also fast to start and ultra-portable. Moving container workloads from your laptop, to the cloud, and then to VMs or bare metal in your data center is a breeze.

Linux Containers:

- Modern containers started in the Linux world and are the product of an immense amount of work from a wide variety of people over a long period of time. Just as one example, Google Inc. has contributed many related technologies to the Linux kernel. Without these, and other contributions, we wouldn't have modern containers today.
- Some of the major technologies that enabled the massive growth of containers in recent years include kernel namespaces, control groups, and of course Docker. To re-emphasize what was said earlier - the modern container ecosystem is deeply indebted to the many individuals and organizations that laid the strong foundations that we currently build on!
- Despite all of this, containers remained complex and outside of the reach of most organizations. It wasn't until Docker came along that containers were effectively democratized and accessible to the masses.
- There are many operating system virtualization technologies similar to containers that pre-date Docker and modern containers.
- Some even date back to System/360 on the Mainframe. BSD Jails and Solaris Zones are some other well-known examples of Unix-type container technologies.
- However, in this section we are restricting our conversation and comments to modern containers that have been made popular by Docker.

Hello Docker!

- Docker is the magic that made Linux containers usable for mere mortals.
- Put another way, Docker, Inc. made containers simple!

Windows Containers:

- Over the past few years, Microsoft Corp. has worked extremely hard to bring Docker and container technologies to the Windows platform.
- At the time of writing, Windows containers are available on the Windows 10 and Windows Server 2016 platforms. In achieving this, Microsoft has worked closely with Docker, Inc.Windows Containers
- The core Windows technologies required to implement containers are collectively referred to as Windows Containers.
- The user-space tooling to work with these Windows Containers is Docker.
- This makes the Docker experience on Windows almost exactly the same as Docker on Linux. This way developers and sysadmins familiar with the Docker toolset from the Linux platform will feel at home using Windows

Windows Containers vs Linux Containers:

- It’s vital to understand that a running container uses the kernel of the host machine it is running on.
- This means that a container designed to run on a host with a Windows kernel will not run on a Linux host.
- This means that you can think of it like this at a high level – Windows containers require a Windows Host, and Linux containers require a Linux host. However, it’s not that simple…
- It is possible to run Linux containers on Windows machines.
- For example, Docker for Windows (a product offering from Docker, Inc. designed for Windows 10) can switch modes between Windows containers and Linux containers. This is an area that is developing fast and you may consult the Docker documentation for the latest updates.

What About Mac Containers?

- There is currently no such thing as Mac containers.
- However, you can run Linux containers on your Mac using the Docker for Mac product. This works by seamlessly running your containers inside of a lightweight Linux VM running on your Mac. It’s extremely popular with developers who can easily develop and test their Linux containers on their Mac.

Docker:

When somebody says “Docker” they can be referring to any of at least three things:

1. Docker, Inc. the company
2. Docker the container runtime and orchestration technology
3. Docker the open source project (this is now called Moby)

Docker - The TLDR:

- Docker is software that runs on Linux and Windows. It creates, manages and orchestrates containers.
- The software is developed in the open as part of the Moby open-source project on GitHub.
- Docker, Inc. is a company based out of San Francisco and is the overall maintainer of the open-source project.
- Docker, Inc. also has offers commercial versions of Docker with support contracts etc.

Docker, Inc.

- Today Docker, Inc. is widely recognized as an innovative technology company with a market valuation said to be in the region of $1.3 Billion. At the time of writing, it has raised over $180M via 7 rounds of funding from some of the biggest names in Silicon Valley venture capital. Almost all of this funding was raised after the company pivoted to become Docker, Inc.
- Since becoming Docker, Inc. they’ve made several small acquisitions, for undisclosed fees, to help grow their portfolio of products and services.

The Docker Runtime and Orchestration Engine:

- When most technologists talk about Docker, they're referring to the Docker Engine.
- The Docker Engine is the infrastructure plumbing software that runs and orchestrates containers. If you're a VMware admin, you can think of it as being similar to ESXi. In the same way that ESXi is the core hypervisor technology that runs virtual machines, the Docker Engine is the core container runtime that runs containers.
- The Docker Engine can be downloaded from the Docker website or built from source from GitHub. It's available on Linux and Windows, with open-source and commercially supported offerings.
- At the time of writing there two main editions:

    1.Enterprise Edition (EE)

    2.Community Edition (CE)

Written in Golang :

- Most of the project and its tools are written in Golang - the relatively new system-level programming language from Google also known as Go. If you code in Go you're in a great position to contribute to the project!

The Container Ecosystem:

- One of the core philosophies at Docker, Inc. is often referred to as Batteries included but removable.
- This is a way of saying you can swap out a lot of the native Docker stuff and replace it with stuff from 3rd parties. A good example of this is the networking stack. The core Docker product ships with built-in networking.
- But the networking stack is pluggable meaning you can rip out the native Docker networking and replace it with something else from a 3rd party. Plenty of people do that.
- In the early days it was common for 3rd party plugins to be better than the native offerings that shipped with Docker. However, this presented some business model challenges for Docker, Inc. After all, Docker, Inc. has to turn a profit at some point to be a viable long-term business. As a result, the batteries that are included are getting better and better. This is causing tension and raising levels competition within the ecosystem.

- To cut a long story short, the native Docker batteries are still removable, there's just less and less reason to need to remove them.

The Open Container Initiative (OCI):

- No discussion of Docker and the container ecosystem is complete without mentioning the Open Containers Initiative - OCI.
- The OCI is a relatively new governance council responsible for standardizing the most fundamental components of container infrastructure such as image format and container runtime.
- From day one, use of Docker has grown like crazy. More and more people used it in more and more ways for more and more things. So, it was inevitable that somebody was going to get frustrated. This is normal and healthy.
- A company called CoreOS didn't like the way Docker did certain things. So they did something about it! They created a new open standard called appc that defined things like image format and container runtime. They also created an implementation of the spec called rkt (pronounced "rocket").
- This put the container ecosystem in an awkward position with two competing standards.
- Getting back to the story though, this threatened to fracture the ecosystem and present users and customers with a dilemma. While competition is usually a good thing, competing standards is usually not. They causeconfusion and slowdown user adoption. Not good for anybody.
- With this in mind, everybody did their best to act like adults and came together to form the OCI - a lightweight agile council to govern container standards.
- At the time of writing, the OCI has published two specifications (standards):

  1.The image-spec

  2.The runtime-spec

The Problem With Two Standards:

- An analogy that's often used when referring to these two standards is rail tracks.
- These two standards are like agreeing on standard sizes and properties of rail tracks. Leaving everyone else free to build better trains, better carriages, better signalling systems, better stations... all safe in the knowledge that they'll work on the standardized tracks. Nobody wants tw competing standards for rail track sizes!

The OCI:

- So far, the OCI has achieved good things and gone some way to bringing theecosystem together.
- However, standards always slow innovation! Especially with new technologies that are developing at close to warp speed.
- This has resulted in some passionate discussions in the container community.
- The OCI is organized under the auspices of the Linux Foundation and both Docker, Inc. and CoreOS, Inc. are major contributors.

Installing Docker:

- There are loads of ways and places to install Docker.
- There's Windows, there's Mac, and there's obviously Linux.
- But there's also in the cloud, on premises, on your laptop.
- Not to mention manual installs, scripted installs, wizard-based installs.
- There literally are loads of ways and places to install Docker!

Linux:

sudo apt-get update

sudo apt-get install docker.io

Dev / Ops Perspectives:

The Ops Perspective:

- download an image.
- start a new container.
- log in to the new container.
- run a command inside of it.
- destroy container.

The Dev Perspective:

- pull some app-code from GitHub.
- inspect a Docker file.
- containerize the app.
- run app as a container.

2 Major Components:

When you install Docker, you get two major components:

- The Docker client
- The Docker daemon (sometimes called “server” or “engine”)

You can test that the client and daemon are running and can talk to each other with the docker version command.adil:/ adil$ docker version

Client:

Version: 18.06.1-ce

API version: 1.38

Go version: go1.10.3

Git commit: e68fc7a

Built: Tue Aug 21 17:21:31 2018

OS/Arch: darwin/amd64

Experimental: false

Server:

Engine:

Version: 18.06.1-ce

API version: 1.38 (minimum version 1.12)

Go version: go1.10.3

Git commit: e68fc7a

Built: Tue Aug 21 17:29:02 2018

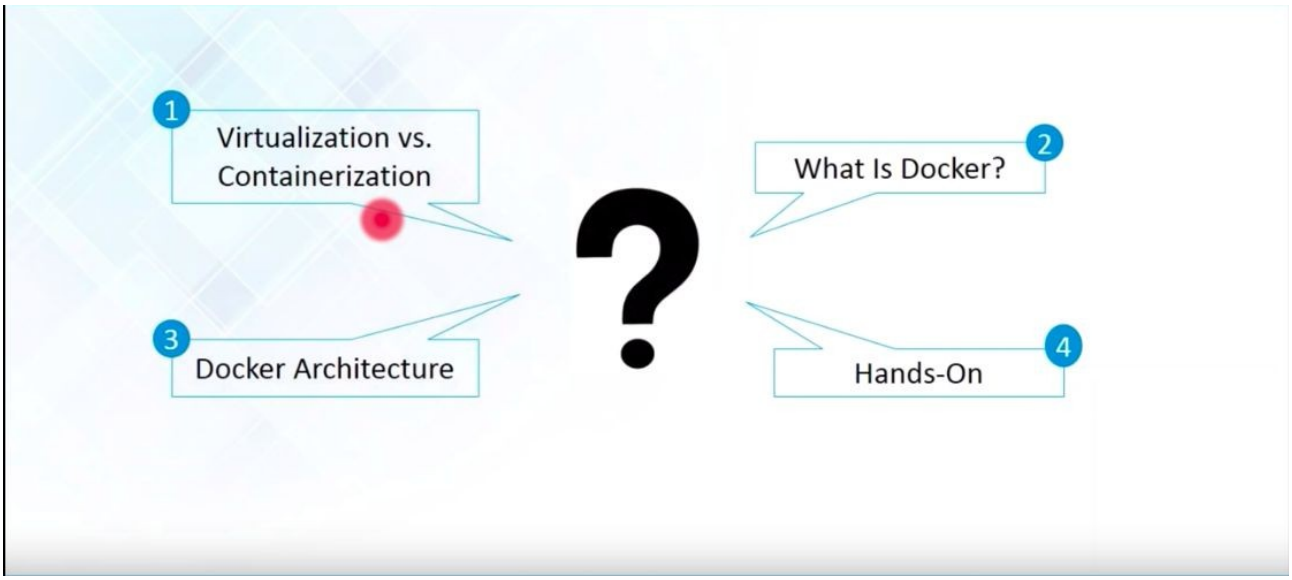
OS/Arch: linux/amd64

Doesn’t work?

- If you get a response back from the Client and Server components you should be good to go.
- If you are using Linux and get an error response from the Server component, try the command again with sudo in front of it:

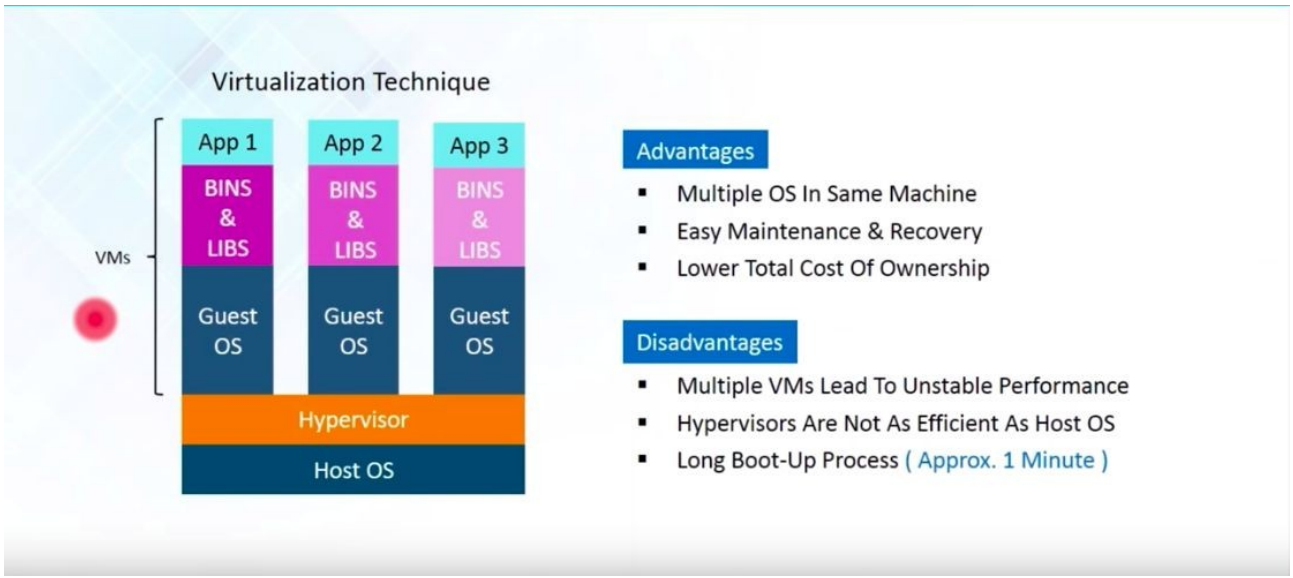
  sudo docker version

- If it works with sudo you will need to add your user account to the local docker group, or prefix the remainder of the commands in this chapter with sudo.

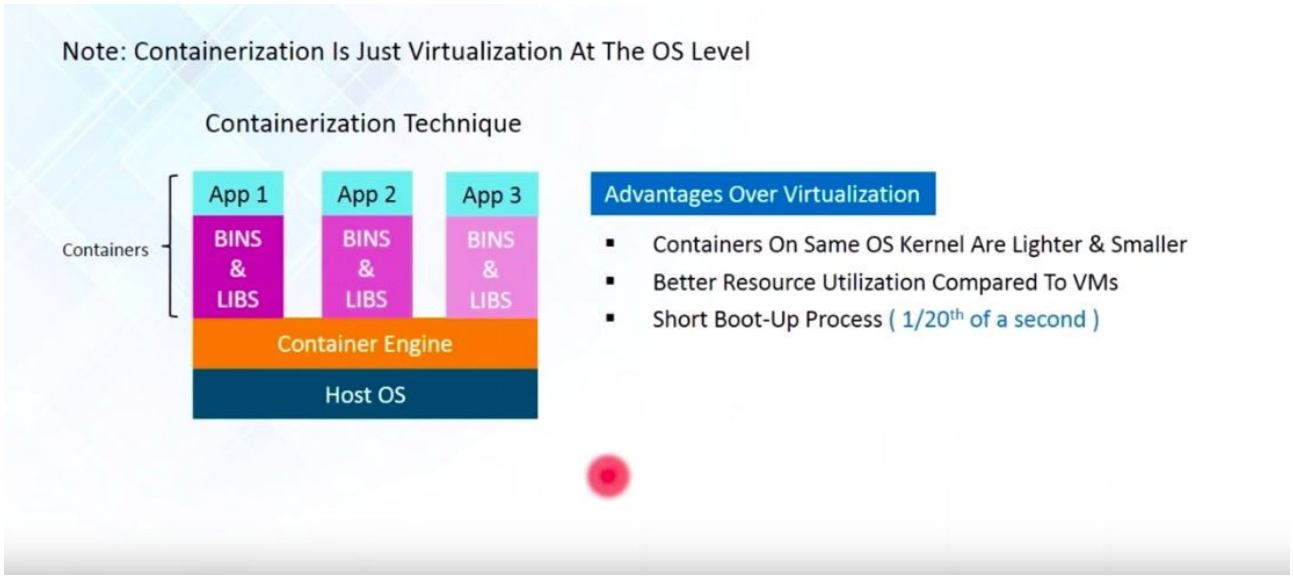


Virtualization & Containerization

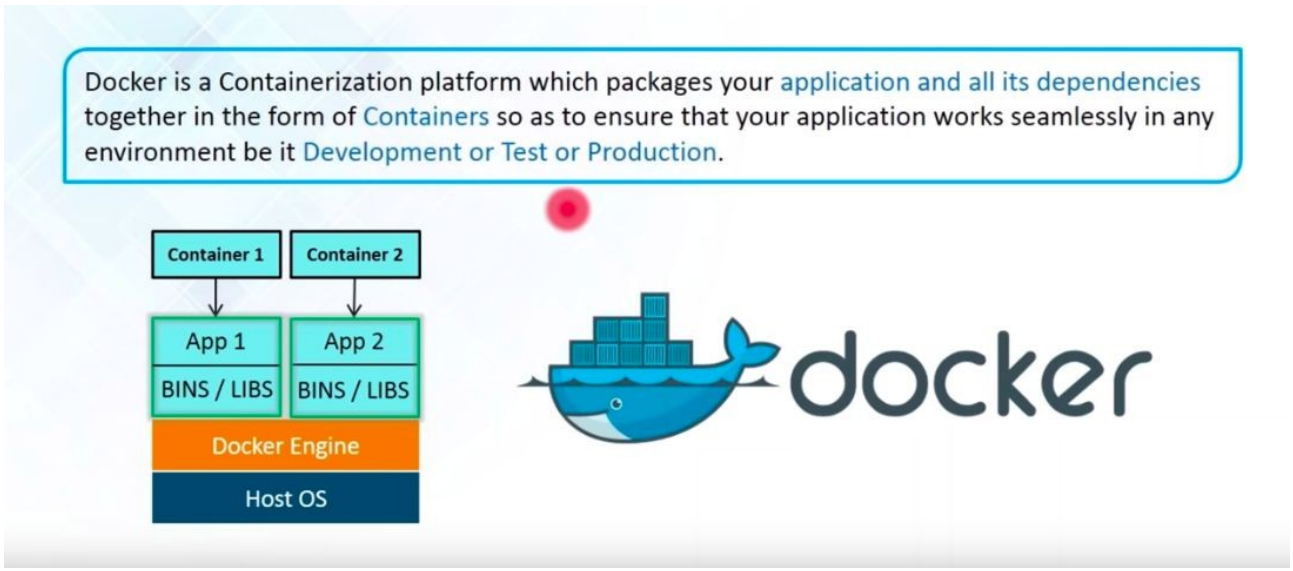
Virtualization: Adopted by VMs



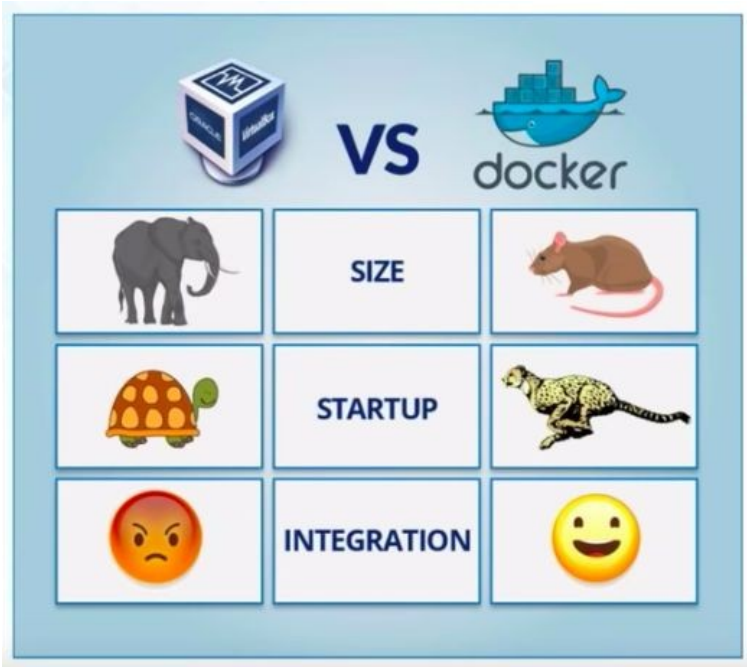
Containerization



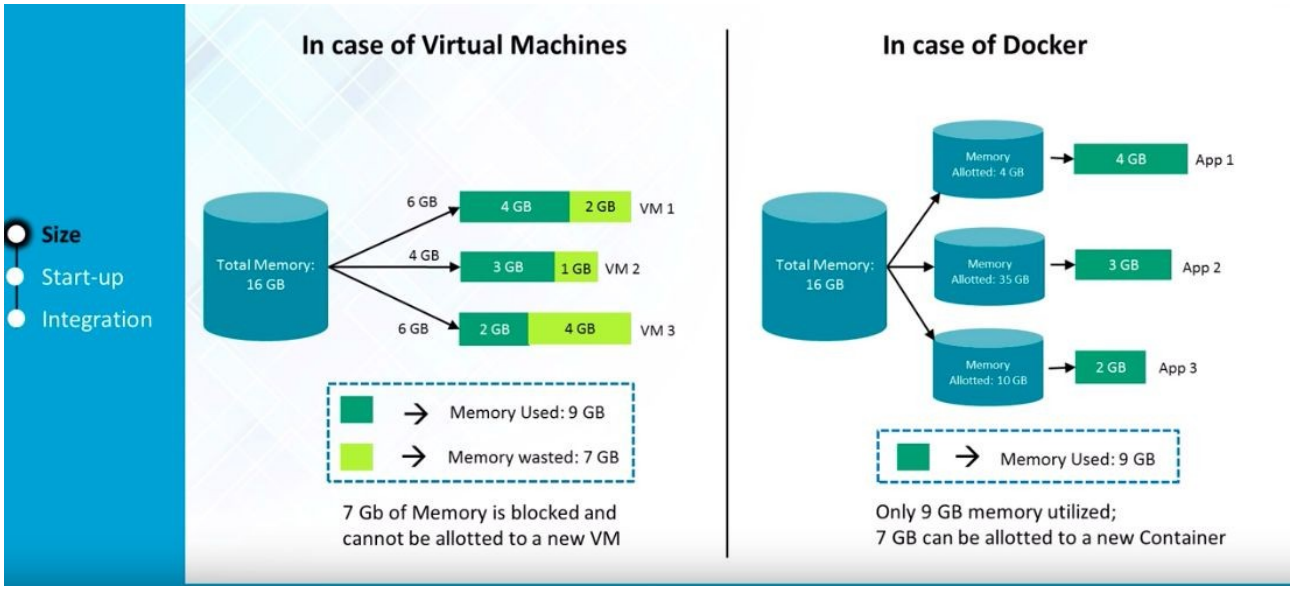
Docker



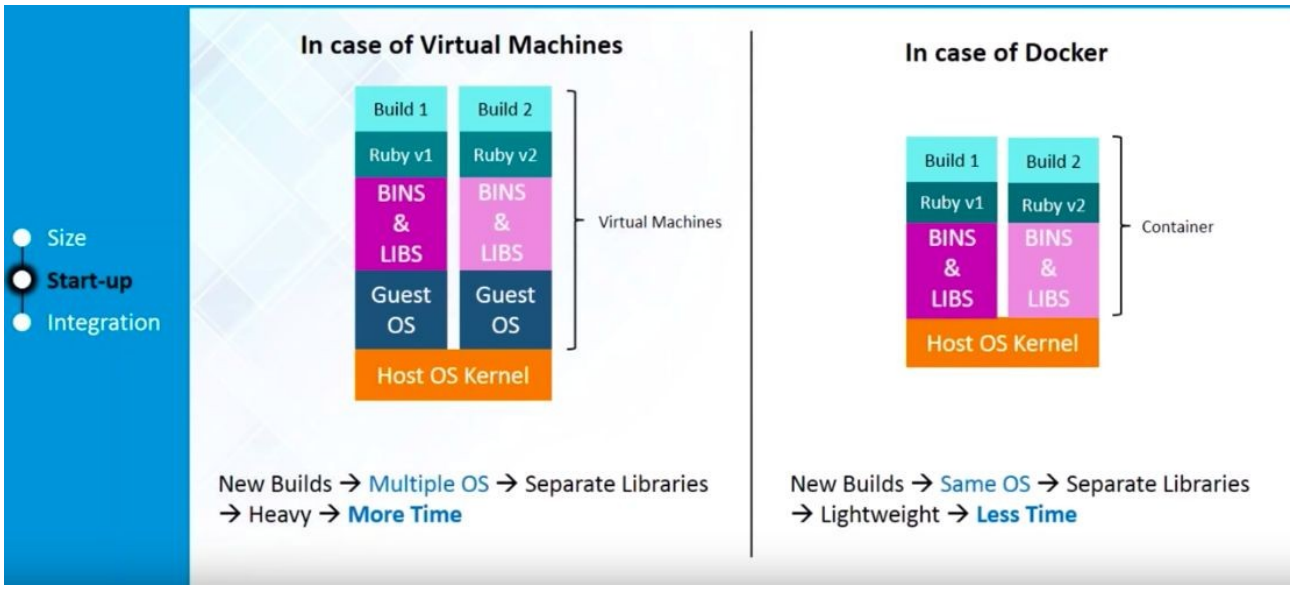
VM vs. Docker



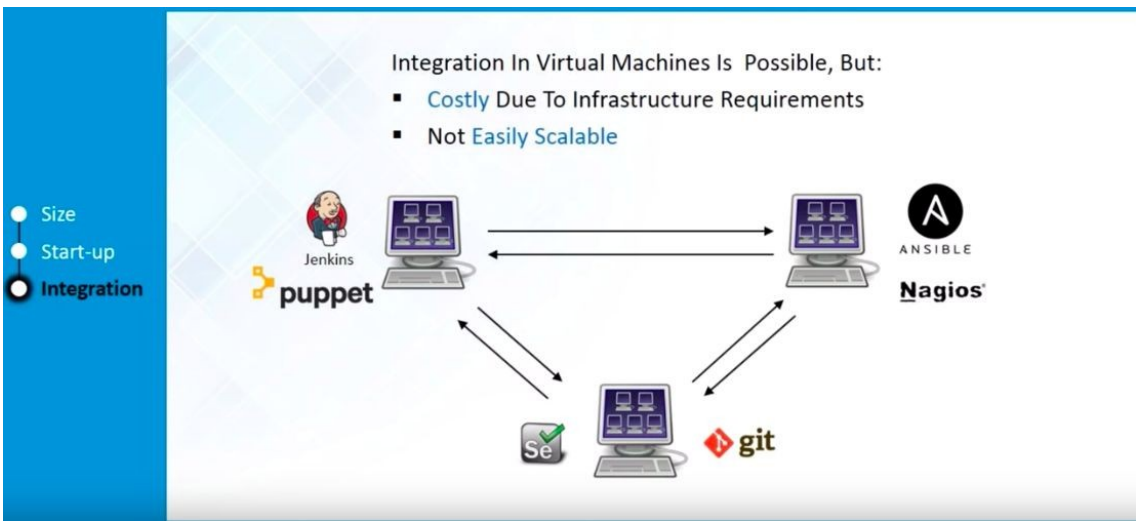
Resources / Memory Utilization



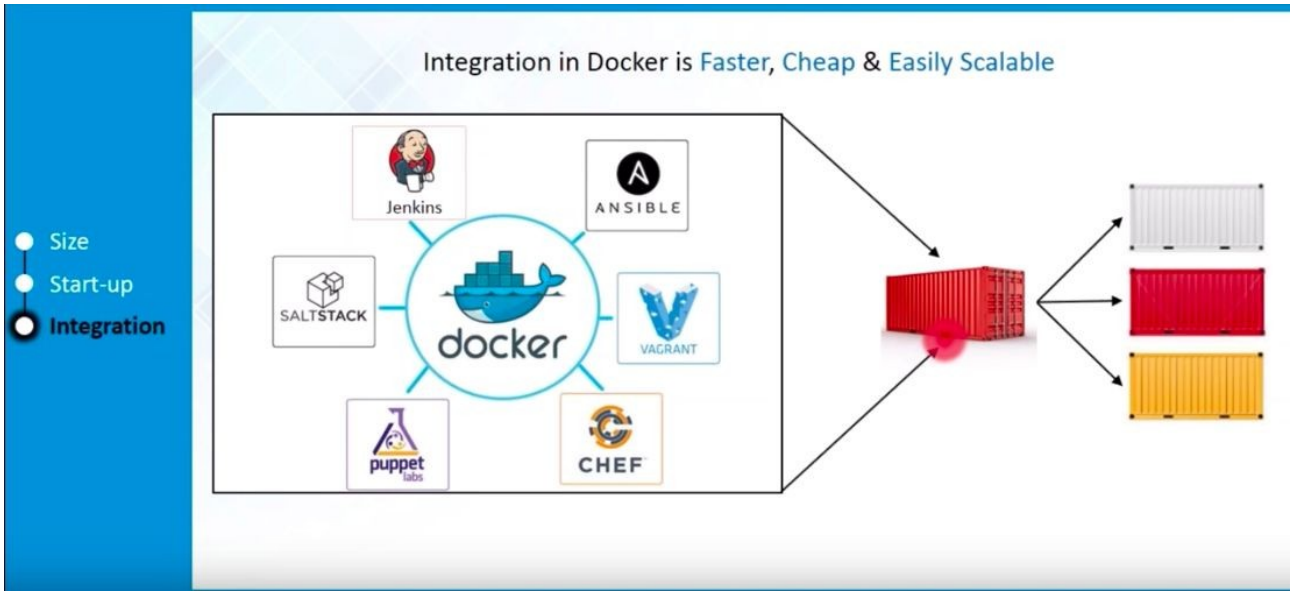
Building and Deployment



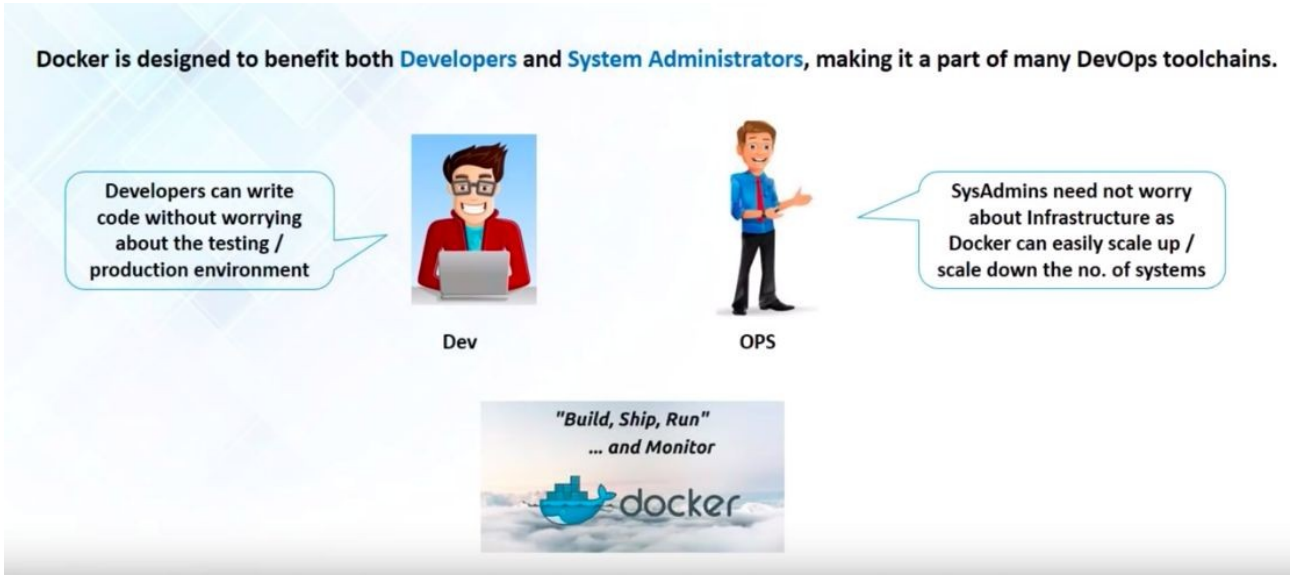
Integration in VMs



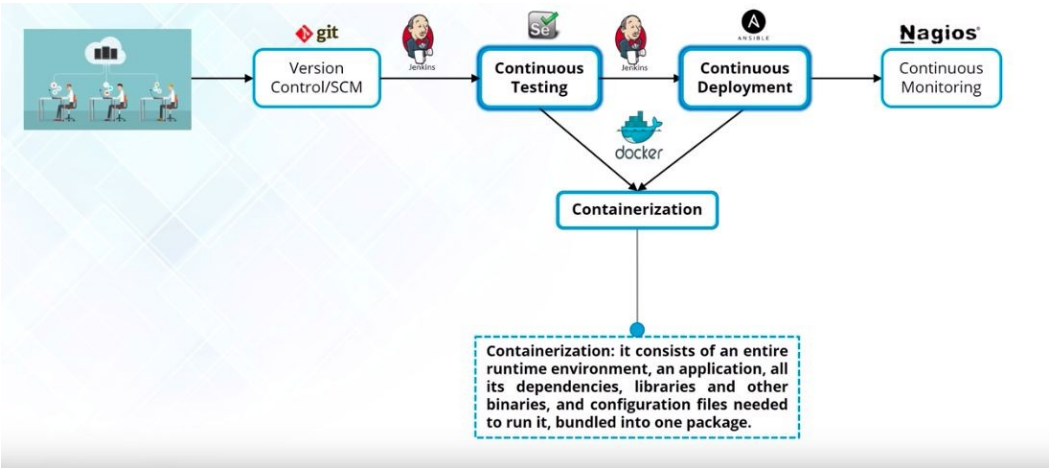
Docker - DevOps



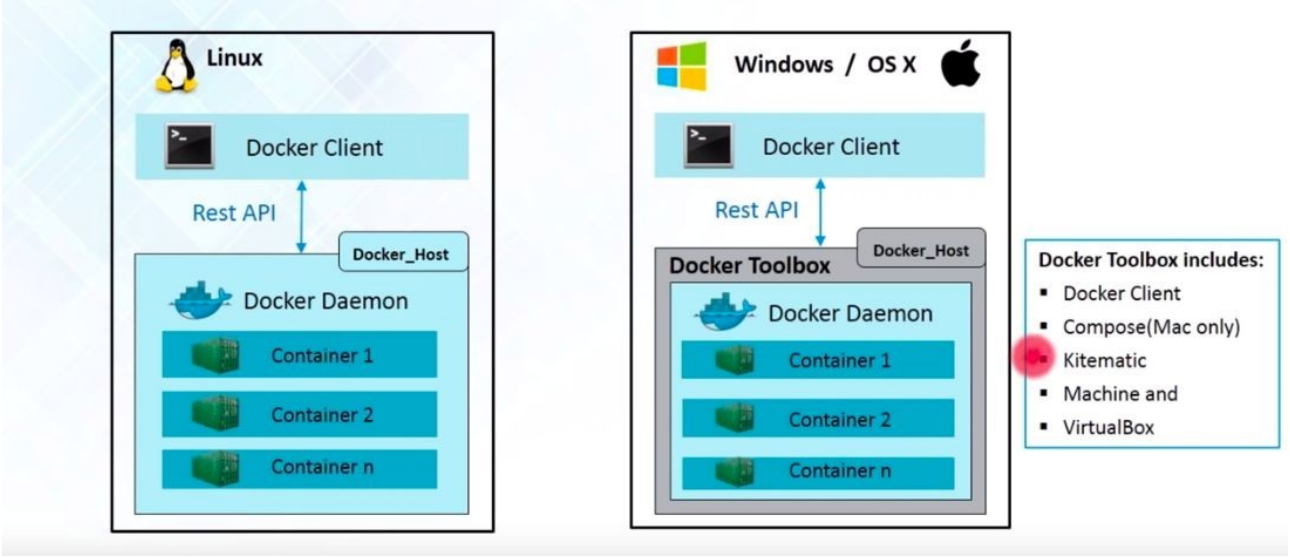
Who can use Docker



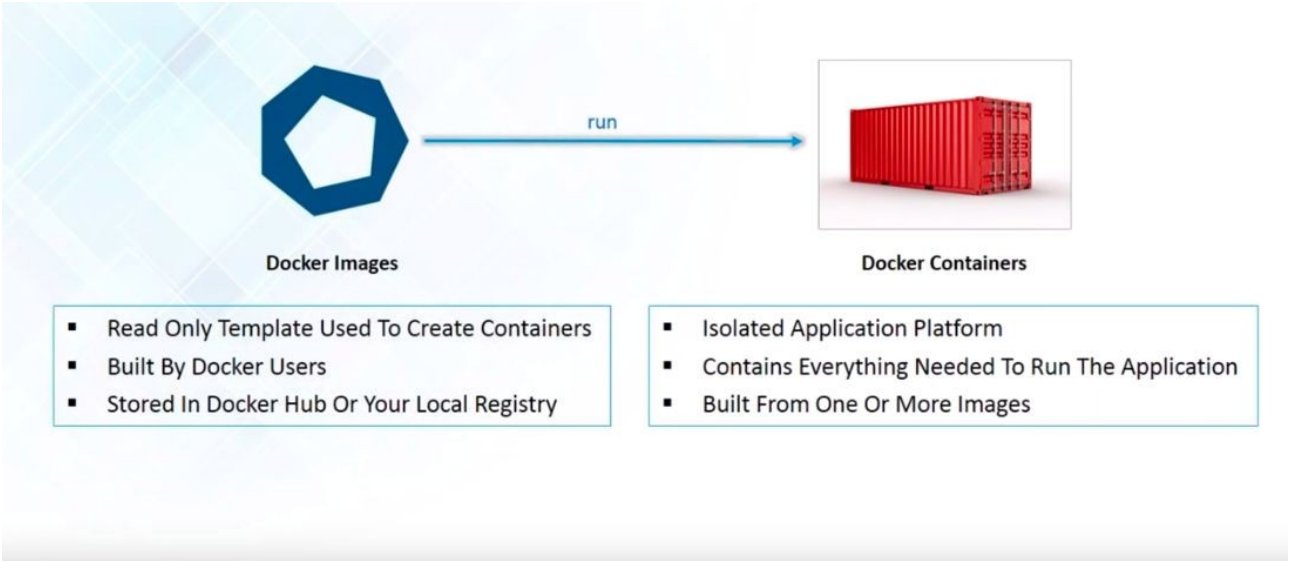
How Docker used in DevOps



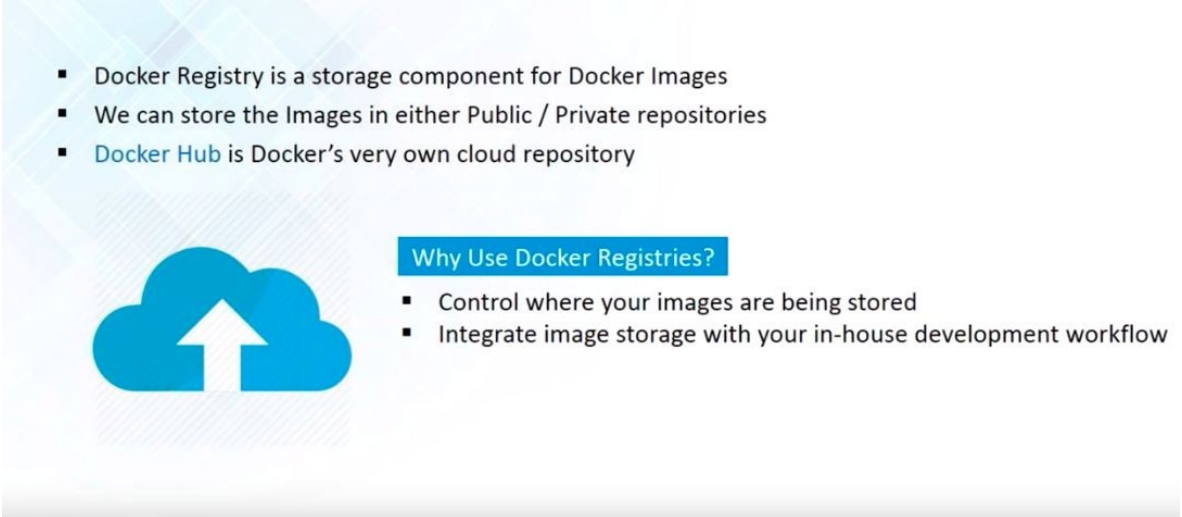
Docker Engine



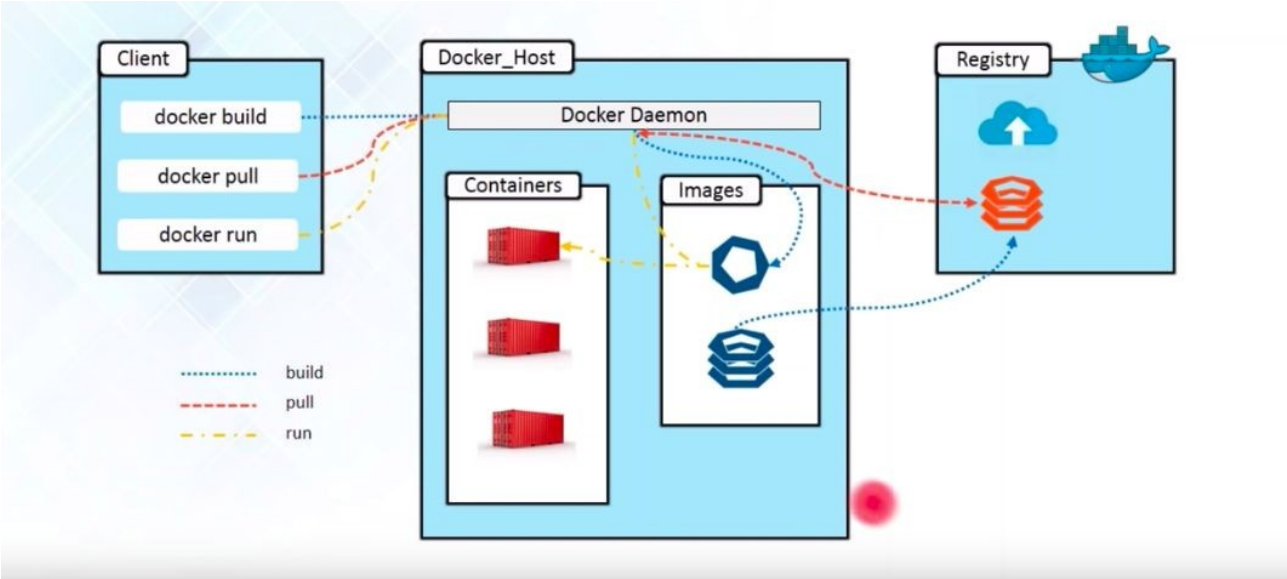
Docker Images & Containers



Docker Registry



Docker Architecture



Basic Docker Commands

To Pull a Docker image from the Docker hub, we can use the command:
$ docker pull <image-name:tag>

To Run that image, we can use the command:
$ docker run <image-name:tag> or **$ docker run <image-id>**

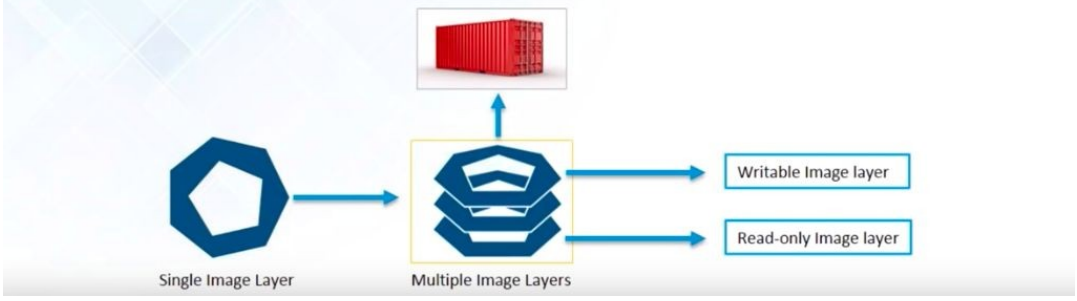
To list down all the images in our system, we can give the command:
$ docker images

To list down all the running containers, we can use the command:
$ docker ps

To list down all the containers (even if they are not running), we can use the command:
$ docker ps -a

Building Images

- Images are comprised of multiple layers
- Each layer in an Image is an Image of its own
- They comprise of a Base Image layer which is read-only
- Any changes made to an Image are saved as layers on top of the Base Image layer
- Containers are generated by running the Image layers which are stacked one above the other



Creating a Dockerfile

Docker File

It contains instructions for building a Docker Image. Instructions contain various keywords.

FROM → This keyword indicates the base image from which the container is built

RUN → This keyword indicates the command that needs to be executed on the Image

Command to Build an Image:

$ docker build –t [image_name]:tag .

$ docker build –t edureka:1.0 .

Command to Run the newly built Image:

$ docker run --name "container-name" -p <host port>:<container port> <image_name:tag>

$ docker run --name "ubuntutomcat7" –p 8080:8080 edureka:1.0

Docker file

Dockerfile ×

```
FROM ubuntu

# ADD JAVA repo
RUN apt-get update && apt-get install -y curl \
  python-software-properties \
  software-properties-common \
  && add-apt-repository ppa:webupd8team/java

# Installa Java
RUN echo debconf shared/accepted-oracle-license-v1-1 select true | debconf-set-selections \
 && echo debconf shared/accepted-oracle-license-v1-1 seen true | debconf-set-selections \
 && apt-get update && apt-get -y install oracle-java7-installer

#  Install Tomcat
RUN mkdir -p /opt/tomcat \
 && curl -SL http://apache.fastbull.org/tomcat/tomcat-7/v7.0.72/bin/apache-tomcat-7.0.72.tar.gz \
 | tar -xzC /opt/tomcat --strip-components=1 \
 && rm -Rf /opt/tomcat/webapps/docs /opt/tomcat/webapps/examples

COPY tomcat-users.xml /opt/tomcat/conf/

# Expose tomcat
EXPOSE 8080

ENV JAVA_OPTS -server -XX:+DisableExplicitGC -XX:+UseConcMarkSweepGC \
  -Xms1G -Xmx2G -XX:PermSize=1G -XX:MaxPermSize=2G

WORKDIR /opt/tomcat
CMD ["bin/catalina.sh","run"]
```

Images:

- A Docker image is as an object that contains an OS file system and an application.
- An image is effectively a stopped container.

  $ docker image ls
- If you are working from a freshly installed Docker host it will have no images.

Pulling Images:

- Getting images onto your Docker host is called "pulling"
- $ docker image pull alpine:latest