Hello folks, I hope you are doing great.

In this video, let’s try to answer 3 questions.

- What is docker and it’s analogy in the real world
- Why docker
- How docker

So let’s imagine **Ira** is a fresh computer science graduate. After graduation, she also got a good job in her hometown. And she is very happy about it.

Let's imagine today is her first day to join the new office. Everybody at her home is excited as well. So her mom packs her favourite lunch in newly bought lunchbox. Taking lunch boxes to the office is quite common in India, and maybe too many other countries. Saves tons of money.

Anyway, let’s get back to Ira taking her lunch box to the office. She attends the orientation session held by the HR team. She meets her colleagues and makes new friends. Now, it’s already time for lunch. Ira opens up her lunch box and gets shocked to see that sandwich, chapati/roti and curry is already having a party inside the lunchbox. It’s difficult to look at that mess let alone eat it. She was looking forward to eating her favourite food but things turned out to be different. She is feeling angry but can’t really help it now. She grabs something to eat from the office canteen and goes back to her work.

She comes home and narrates the incident to her parents.

Now, her mom is thinking about what could be done about this problem. Definitely the flat lunch box is not helping. So her mom buys another tiffin box from the market.

This tiffin box has proper containers to keep each item separate from every other item. Curry does not mix with chapati and finally Ira can enjoy her favourite food at the office.

I want you to remember this short story and we are going to revisit parts of it when we talk about docker.

In simple terms, the first flat tiffin box can be considered as a monolithic application, which had created a mess. We will talk more on what is a monolithic application in detail later on or I’ll create a separate video and link it here.
Next, **Ira’s** mom bought a new tiffin which had separate containers to store each food item. Those individual containers are nothing but docker containers which are trusted to handle only one thing. Imagine 3 docker containers coming together to serve you best lunch or the great app that you want to build.
Each lunch box container was packed with one item. But each item has its own complexity for e.g. sandwich had cheese within the bread, tomato slices, cucumber slices and what not. Similarly a docker container will be responsible for only one thing like auth service but within auth it will have many different components, libraries etc

I hope on a high level, you understood what docker stands for, how they should be used and what is their main purpose.

Let's now discuss why docker.
To some extent we did speak about why docker when we briefly mentioned monolithic application and how it creates a mess when not handled well.
Let's do a deep dive here. Why do we need docker for?

1. If you have not worked on docker before or development does not happen on docker then you might have come across a situation where you see something working on your machine but not your colleagues machine. Such issues are often reported by the QA team that certain functionality is not working as expected and when you check it, it seems to be working fine on your machine. The difference could be the underlying hardware or set of tools which are required to run the application. It could also update patch versions of specific OS.
   With docker these issues get completely eliminated. I've another analogy here. We know C or C++ is platform dependent whereas Java is called platform independent. Which means compile once and run anywhere, be it windows machine or mac or linux based os. Do you know why it works because all the machines have a JVM which runs the bytecode and does the translation as per the host machine.

   Similarly, we have a docker daemon when it is installed it will make sure your application runs the same way on multiple platforms. Docker daemon does not heavy lifting under the hood.

   Also, since you pack everything under the docker container, your environment never changes no matter who instals it or when it gets installed. If you pack your docker image with node 17 then it remains the same across all its installations, be it dev env or qa env or dev machine or qa machine. I hope that clears the importance of how docker helps in these situations.

2. In the current world we want everything online, our grocery is online, our shopping is online, money transfer is online and it's an integral part of our lives. Because of this traffic for any services has gone up by many folds. Compute requirements, storage requirements, everything has gone through the roof. Each service has different compute power needs. For e.g. you may have 1000 users signing up per hour but you may have 10000 users browsing the shopping catalogue. So you need more computation power for shopping catalogue than auth service. So we need an ability to be able to scale independently. With monolithic apps this becomes a challenge where you can not scale a specific module. In this use case, docker container with specific purpose like a docker container with auth service, a docker container with shopping catalogue can be scaled independently based on the need.
3. Another advantage that you get with docker is flexibility? How? Let's talk.
   Because there is a separate container for each service, then you can choose to write each service based on your needs. These needs could be your team's experience in a specific programming language. For e.g. you can auth service in nodejs, and shopping catalogue service which needs to handle high concurrency then you can choose to write that specific service in go language to support the concurrency need.

This also helps you revamping your infrastructure in future, what does that mean is. There is another programming language which runs at the speed of light, everybody is talking about it. Then out of 50 services that you have you can choose to experiment and re-write that service in a new programming language. This way you can reduce the risk, you also reduce tech debt and it happens in step by step fashion without stopping any services. This way you continue to improve your platform.

I think we talked enough on what, why of the docker, now let’s talk about how of the docker. How does docker work under the hood?

In order to understand docker, let’s understand what virtualization is. In layman's terms it is a computer within a computer. That’s a very basic picture to visualise the concept of virtualization.

Imagine you have a great computer 30 TB harddisk, 128 CPU cores, 248 GB RAM. The cost of such computers is very high and if you don’t utilise it to its fullest then it will be a waste of money.
Using the concept of virtualization, we can have multiple VM’s in this computer. Say, 5 machines, with the configuration of 4 TB harddisk, 32GB Ram, 16 CPU cores. And multiple people can use these VM’s without affecting the other VM’s. Meaning, once you have been allocated the specific configuration then it is reserved for you physically even if it is underused or even if it’s overused but you will not get more disks or ram than what is allocated to you. This is the overall use of virtualization. And this is how your ec2 machines work in the cloud.

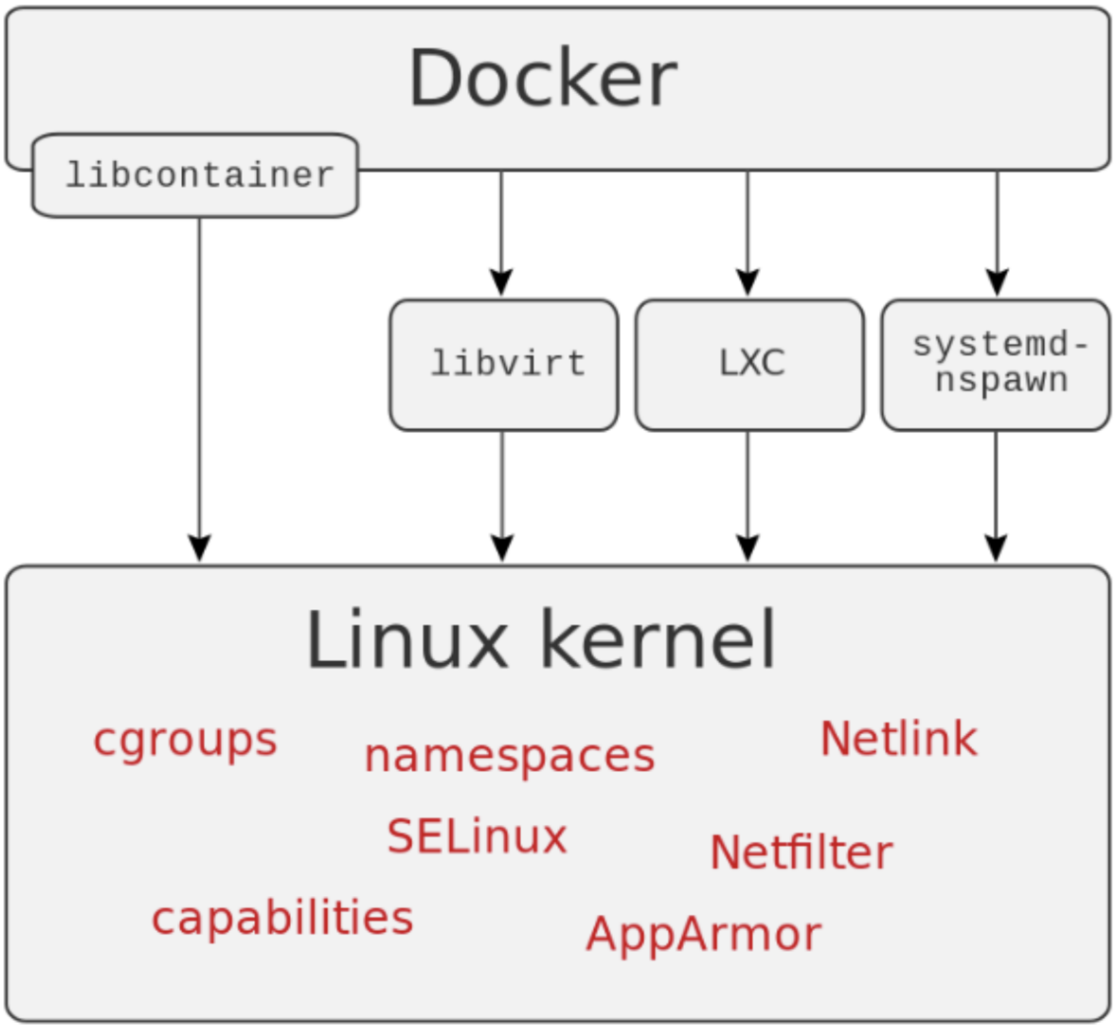
Then how different is docker and how does it work.

The main difference between how docker and VM technology work is that when a VM is installed the entire OS is installed along with it but docker uses the host machine OS that’s why it is lightweight. The docker daemon is responsible which acts as middlemen between the host machine and docker container.

The main components which helps in making docker work is cgroups and namespaces. . With cgroups, the Linux operating system can easily manage and monitor resource allocation for a given process and set resource limits, like CPU, memory, and network limits.

Namespaces are helpful in isolating process groups from each other. There are six default namespaces in Linux: mnt, IPC, net, usr, pid, and uts. Each container will have its own namespace and processes running inside that namespace, and will not have access to anything outside its namespace.

Docker containers also have network isolation (via libnetwork), allowing for separate virtual interfaces and IP addressing between containers.


Docker
libcontainer
libvirt
LXC
systemd-nspawn
Linux kernel
cgroups
namespaces
Netlink
SELinux
Netfilter
capabilities
AppArmor