

IIT Madras
ONLINE DEGREE

Modern Application Development-I
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Containers

Hello, everyone, and welcome to this course on modern application development.

(Refer Slide Time: 00:15)

IIT Madras
BS4 Degree

Containers

- What?
 - self-contained environment with OS and minimal libraries - just enough to run process
 - Primarily used with Linux kernel namespaces, others like chroot possible
- Why?
 - Full OS impossible to version control - too much software, too many versions
 - Create self-contained images that can be version controlled
 - Sandboxing - image cannot affect other processes on system
- How?
 - Kernel level support needed
 - All communication "inter-container" - networking

So, now finally, a topic which is, in some sense, separate from the things that we have talked about earlier. But is also in some ways related because it facilitates the CI, CD kind of approach. It is something called containers, which is once again, a word that you probably come across a lot. You might have heard of the term docker, very likely heard the term Kubernetes and they are all essentially examples of containers in action.

So, the first question is what is a container? And a good way of describing it is that it is a self contained environment, with an operating system, and minimal set of libraries just enough to run some process. And the interesting thing about this is, you could think of, if I had a virtual machine, after all, that is a container.

And yes, technically, that is a container. But that is not really the sense in which the word is used. After all, a virtual machine, you can install your own OS and your libraries and just install whatever you need in order to get something running. But that is typically not what we mean

by a container, a container is supposed to have less sort of impact on the resources of the system. And the way that is achieved is, it is primarily used with something like the Linux kernel, and the Linux kernel has a facility called control groups, where they define something called namespaces, which means that each process has a namespace associated with itself. What that namespace means is, when it refers to any system calls or library functions, or trying to access the file system or any other resource, the operating system knows it is coming from this particular process. And it can prevent this process from affecting any other processes.

So, by making use of these namespaces, what you can do is? A full operating system is pretty much impossible to version control, meaning that if I say that, you need to install a complete operating system with ubuntu 20.04, this version of the apache web server, this version of Python, this version of the interface, etc,etc. You can do it. But it is quite complicated, because there are so many moving parts, so many versions of different pieces of software.

Instead, if you could sort of say that I will just package the minimal libraries and a web server, and say that now this particular container is the web server, it has everything you need associated with it, you do not need to upgrade the OS. You do not need to install any new software, you do not need any other libraries, it has everything it needs by itself. It makes it a bit easier to sort of control the versions and share among different people. Which means that you can essentially create these kinds of self contained images, which have all the libraries they need.

And one other very important part of why you would want to do this kind of containers is what I said earlier, that one process cannot sort of interfere with another process. That is called Sandboxing. Each process is sort of given its own sandbox, you make a mess in your sandbox, you and it is like children. You put them in a sandbox, you let them play around with it, you throw the sand up in the air point is nothing gets out of the sandbox.

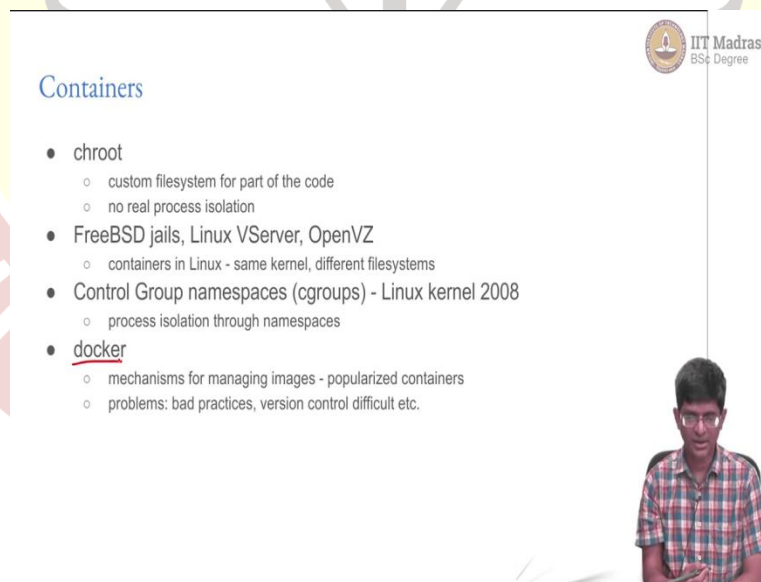
In the same way a process goes and tries to create some files run out of disk space, use up too much memory whatever the container that was given to it had certain restrictions so much memory, so much disk space, so much CPU, you run out of space over there, you are not affecting the rest of the system. So, sandboxing is another important reason why you want to have containers, and the final, so the what and the why are relatively clear. The how is the next important part and over there, you definitely need some support from the operating system kernel.

So, what is operating system kernel, that is the part that directly talks to the hardware and manages resources. So, when we say Linux, usually people talk about Ubuntu and say it is Linux. Now, Ubuntu is a particular distribution of Linux. Linux is the kernel that is running inside the computer. And it is one part of the software of an Ubuntu distribution. You might have the same Linux kernel in an Ubuntu system and in a redhat system, and in maybe, open (())(04:51) system, all of those. You might have the same kernel or you might have slightly different patched versions.

The point is, the kernel is what takes care of Managing Resources, and you need support from the kernel in order to run something like this. You could potentially even have containers with other operating systems like Microsoft Windows, but the most popular ones right now at least are Linux based.

What this also means is that, since each process is now self contained, it cannot directly talk to another processor, it cannot share information with another process. This means all communication between processes is usually in this case done through explicit networking. And you have container level networks that are created for this.

(Refer Slide Time: 05:35)



The slide is titled "Containers" and is part of a presentation from IIT Madras. It lists several container technologies and their features:

- chroot
 - custom filesystem for part of the code
 - no real process isolation
- FreeBSD jails, Linux VServer, OpenVZ
 - containers in Linux - same kernel, different filesystems
- Control Group namespaces (cgroups) - Linux kernel 2008
 - process isolation through namespaces
- docker
 - mechanisms for managing images - popularized containers
 - problems: bad practices, version control difficult etc.

A small video inset in the bottom right corner shows a man in a plaid shirt speaking.

So, there is a long history to containers, they started out with something called chroot, change the root filesystem. Back in the late 70s. Later on, there was this concept in the FreeBSD operating system around early 2000s, where they brought in something called a jail. And they were a

couple of similar projects in Linux as well around that time, openVZ was one, Linux VServer was another.

All of them use different approaches in order to containerize meaning that they try to use the same kernel, but somehow separate out processes from each other. Now, the idea of control group namespaces, was officially introduced into the kernel in 2008. And that allowed process isolation through namespaces. And that is when sort of other this whole containerization really started taking off.

In particular, the popularity started with the introduction of docker, which is one particular software for managing containers. So, docker by itself is not a container does not automatically imply docker. Docker is one way of managing containers. It has lots of mechanisms for managing images. And it really played a great role in popularizing the use of containers.

There are some Gribbs about it in particular, it is sort of easy to fall into certain bad practices in terms of how you do this, there are well defined best practices for using docker. On the other hand, they are not always followed, which means that you can end up with a mess in certain kinds of images that are created.

(Refer Slide Time: 07:11)



Orchestration

- App consists of multiple processes, not just one
- Start in some specific order (dependencies)
- Communicate between processes that are isolated
 - Network
- Mechanisms to build and orchestrate, automate
 - docker-compose
 - Kubernetes
- Key to understanding and managing large scale deployments

IIT Madras
BSc Degree

Now, orchestration is the next step of next phase, once you have the idea of containers. Essentially, what it says is, an app does in consist of a single process, you would typically have a web front end, you would have some kind of thing, which is running all the controllers in your

app, you would have a database. And all of these are separate processes that are running on your system.

So, why not have a separate container for each and use this entire containerization process in order to, go back and build the picture. That we saw right at the start that allows us to scale, have one container that acts as an HTTPS and load balancer front end, have another container that access the actual web front end.

Maybe multiple copies of this have multiple backend, the database containers have another container whose only job is logging. So, rather than having physical servers, they could just be containers, which means that they are more lightweight, they do not take up so much of so many resources on your system, and maybe run hopefully a little bit faster than a full blown virtual machine. But, are easy to start and stop. That is the most important thing.

So, orchestration essentially refers to how you get all of these things to start and work together. It is like you have a big orchestra, lots of different musical instruments, and somebody needs to manage each one and say, now this is when you start this is when you do something else. This is when you need to stop and shut down, you need to start in a certain order, you need to stop in a certain order, all of that is taken care of by something called an orchestration agent.

And over there, there are, docker compose is a relatively simple and easy to use one Kubernetes, on the other hand, is sort of the most popular one right now. It scales in a massive way. I mean, there are deployments of Kubernetes that handle, really large applications. And understanding how it works and using the entire sort of concepts involved over there is key to really building large scale apps that scale out nicely.

So, once again, well beyond the scope of this course, but these are all concepts that you might be interested in moving forward, if you are interested in an app that sort of scales of with size.

(Refer Slide Time: 09:41)



Summary

- App: idea to deployment
 - Requirements - Tests - Code - Integration - Delivery - Deployment
 - Scaling
- Mechanisms
 - HTML + CSS + JS - Frontend user interface
 - Databases, NoSQL, cloud stores - Backend
 - Authentication, proxying, load balancing - "middleware"
 - Platform-as-a-Service - deployment and change management

So, now this is pretty much the final summary slide as far as the course is concerned. You start with an idea. Requirements, come up with tests, write code, integrate the code in different ways, package for delivery, deploy, all of those steps need to be followed. And all of them, you need to know something of the different mechanisms that are available for those in order to get an app that will be used by others.

And a secondary part, which was not really touched upon throughout this course, in some sense was how do you scale? What happened? Why do you need to scale? And how do you scale if you need to? So, what are the mechanisms that we went through as far as the course was concerned, we first saw the front end, you manage it using HTML, CSS, JavaScript.

And the fact that there is a web server, which can talk to a web browser, using the HTTP protocol, which is a well defined and easy to extend and easy to implement protocol. Now, that was the front end. We also talked about databases, different kinds of data storage mechanisms, whether it is something like SQL lite that you used in the various assignments, MySQL, Postgres, MongoDB, various kinds of NoSQL and other kinds of cloud storage systems.

This provides the backend, just the storage of data. On top of that, there are also other things like authentication, load balancing, proxying, logging, all of which constitute something called middleware, because they lie somewhere between the front end and the back end. You want to have authentication before accessing certain data in the backend.

Alright, so, you rather than sort of integrating it directly into the front end, you can sort of add it in as a separate package that somebody else has written for you. And very often these are termed middleware for that reason. And finally, in terms of deploying and getting these systems working in practice, we have platforms as a service.

How do you manage changes, how do you deploy them? How do you do continuous integration? All of these are concepts that you need to know as you sort of try scaling up and building larger applications. So, that is the summary of what we have done overall in this course.

