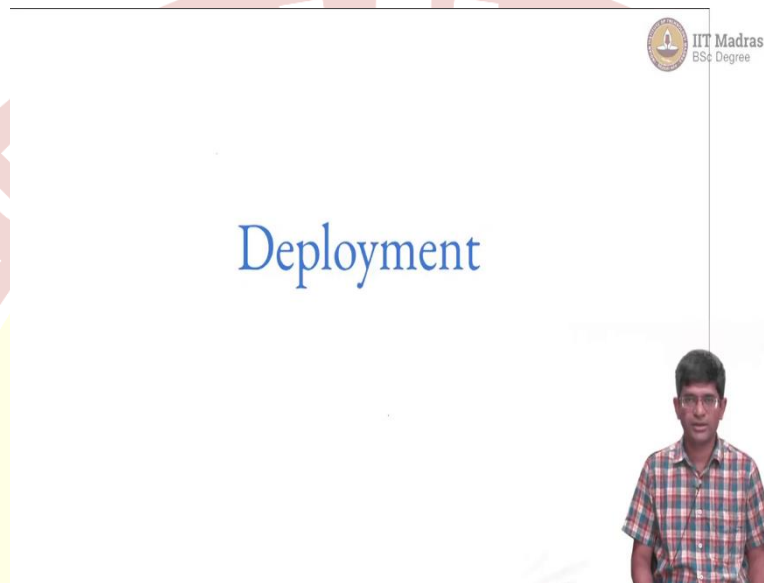


**IIT Madras**  
ONLINE DEGREE

**Modern application development-I**  
**Professor Nitin Chandrachoodan**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Madras**  
**Components of an App**

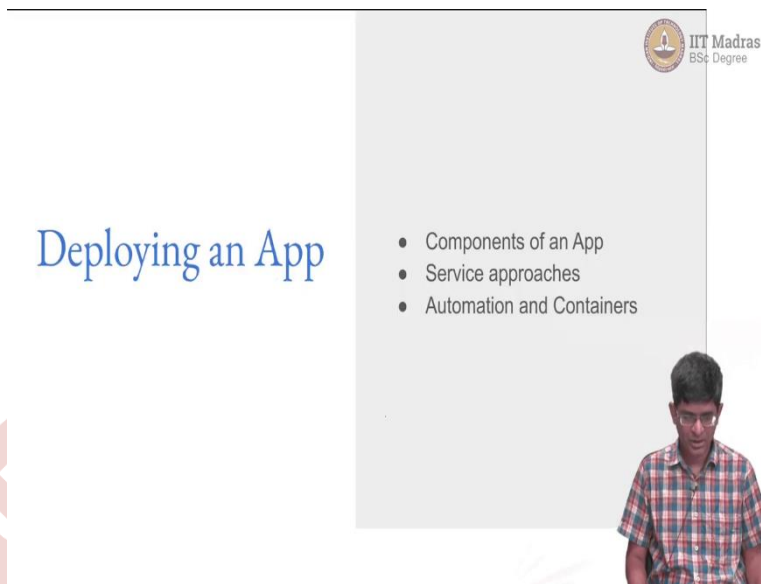
Hello everyone and welcome to this course on Modern Application Development.

(Refer Slide Time: 0:18)



Hello everyone. We are now going to look at one of the topics that sort of brings everything that we have covered in this course together which is essentially the topic of deployment that is you have come up with an app and you want to actually deploy it so that other people can use it. So, what exactly does this mean? What are the steps involved? What are the conditions to think about and what are some ways by which we could go about it? That is what we are going to consider in these few videos.

(Refer Slide Time: 0:42)



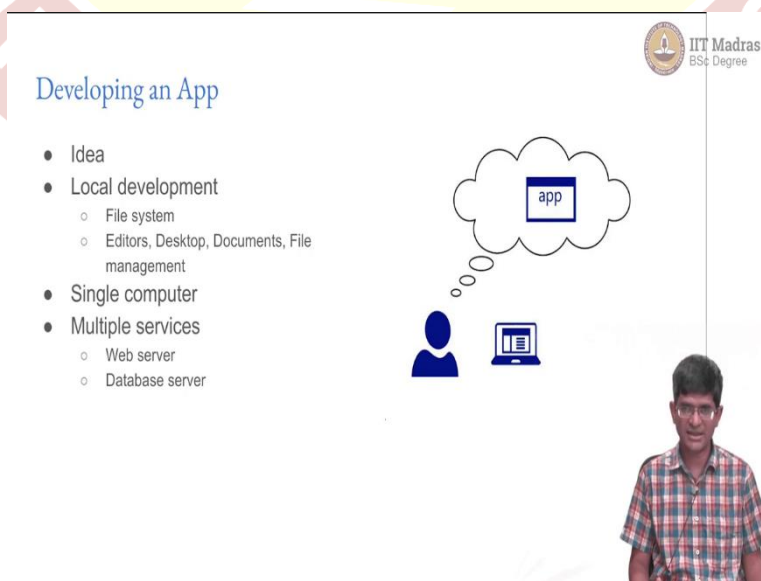
The slide is titled "Deploying an App" in blue text. In the top right corner, there is a logo for "IIT Madras BSc Degree". Below the title, there is a bulleted list of topics: "Components of an App", "Service approaches", and "Automation and Containers". A presenter, a man in a red and blue checkered shirt, is visible in the bottom right corner of the slide frame.

- Components of an App
- Service approaches
- Automation and Containers

So, the first thing that we want to look at is what happens when you want to deploy an app? That is, what are the components of an app? We will start by looking at that. And then, we look at so called service approach which is increasingly common these days. We will define what that is and show you a couple of examples.

And finally look at one very important part of the process of deployment of an app which is how do you automate things? And related topic is related to something called containers which is something that you have probably heard about quite a lot. It is well beyond the scope of this course but is definitely something that you need to be aware of at least. So, we will just sort of touch upon that topic right at the very end.

(Refer Slide Time: 01:23)



The slide is titled "Developing an App" in blue text. In the top right corner, there is a logo for "IIT Madras BSc Degree". Below the title, there is a bulleted list of topics: "Idea", "Local development" (with sub-points: "File system", "Editors, Desktop, Documents, File management"), "Single computer", and "Multiple services" (with sub-points: "Web server", "Database server"). To the right of the list, there is a diagram showing a person icon with a thought bubble above it containing the word "app", and a laptop icon below it. A presenter, a man in a red and blue checkered shirt, is visible in the bottom right corner of the slide frame.

- Idea
- Local development
  - File system
  - Editors, Desktop, Documents, File management
- Single computer
- Multiple services
  - Web server
  - Database server

So, first things first. How do you go about deploying an app? Before deploying an app, you need to have an idea. So, there is a person sitting in front of their laptop and they come up with an idea for an app, maybe it is to make a to-do list, maybe it is to write the next portal for the online degree course, whichever way you think that there is some need for it. People want to interact with something over the web and that you can provide a solution which would be ideally in terms of using the technologies of the web that is to say it is based on a frontend which is accessed through a browser. It essentially uses the HTML, CSS, JavaScript and so on. It has some database connectivity, a few other things of that sort. So, you have an idea to start with

After that you start developing locally. Now, throughout this course we have sort of been saying you can use Replit as one way of developing your code and in fact that is something that I will be coming to a little bit later. It comes to that second topic that I mentioned which is the services. But Replit is still a remote system. What happens if you are suddenly cut off from the internet and you cannot connect to Replit or the Replit servers go down, anything can happen.

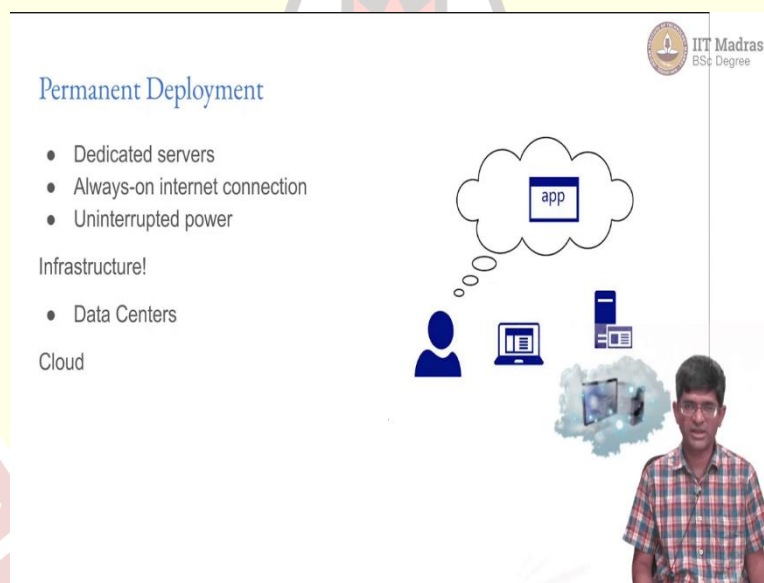
So, in ideal situation, you would want to have all your code locally, which means that you have them in files on your local laptop or desktop, your computer basically and you have some editors that allow you to edit code. You have a desktop that allows to manage files. You have various documents on the files that could include source code. It could include documentation either mark-down files or HTML files or DOCX files or PDFs and you have various other kinds of files images, style files, various things that are required for your app to work.

So, all of them are essentially being developed locally on your computer and that is what we call local development. But all of this is happening on a single machine. It has all the problems associated with that. It means that what if that machine crashes. You could lose everything that you have. But more important you cannot really serve the app for others to use from your local machine. If it is a laptop, you probably want to shut it down at some point and carry it along with you. You go to your university or your office and you want to be able to take the laptop along. That is the whole point. It cannot be always permanently plugged in. It cannot be permanently on either.

And on top of all that you probably have multiple services that are running on this system. It is not just the web frontend. You also have a database server. You might have something, which needs to do some kind of, a different form of processing, some languages running on the machine. All of this might end up becoming too much load for a single person or a single computer.

So, what do you do? This is where, so once you have got up to this stage that is you have developed everything, everything is working locally for you that is when you are ready for deployment and deployment basically means that you wanted to be available for others to use. So, how do you do that? How do you set up the same services that you have already worked with on a remote system in such a way that people can access it, use it, and actually use the app.

(Refer Slide Time: 4:48)



So, this is where we come to what I am going to refer to as permanent deployment. Permanent in the sense that at least it is not dependent on you switching on your computer all the time. You ideally want some kind of a dedicated server something whose job. I mean it is some computer somewhere. It still has to be a computer after all because you developed it on a computer. You want a similar kind of computer to serve the app as well.

But you now you would like it to be something that is dedicated for your tasks, which basically takes care of the work that you want to do. You wanted to have always on internet connection that is of course given because otherwise if somebody tries to reach your app they will not be able to get to it and you want to have uninterrupted power. This is also something

what happens in the data center. You might not be able to guarantee uninterrupted power at home, not just that even if you have a battery backup and so on the connectivity might get disrupted. Something else maybe there is a power outage on your street and the network connectivity goes because of that.

So, all of those are things that you do not really want to deal with, which means that you need some kind of infrastructure and by infrastructure we mean basically the equipment that is handling all of this. You need servers, you need battery backup, you need network connectivity, and it makes sense for people to set up dedicated what they call data centers. So, a data center is essentially a set of computers located in one location, where they specialize in taking care of just the computers alone, make sure that there is uninterrupted power, cooling. You have proper air conditioning in the room, network connectivity with a high bandwidth connectivity to the rest of the world.

All of that is taken care of in a data center and nowadays at least the most sort of prominent way by which we as the general public can get access to data centers is through what we call the cloud. And the cloud essentially is sort of vague concept that sprang up couple of decades ago, where essentially people realize that data centers always existed but they would be dedicated to one company or one university and each company would have its own data center and they would not really share any material or equipment or anything else with others. But over time and got to the point where it did not make sense for every single company to do that, especially small companies do not really want to run through the hassle of having a dedicated data center.

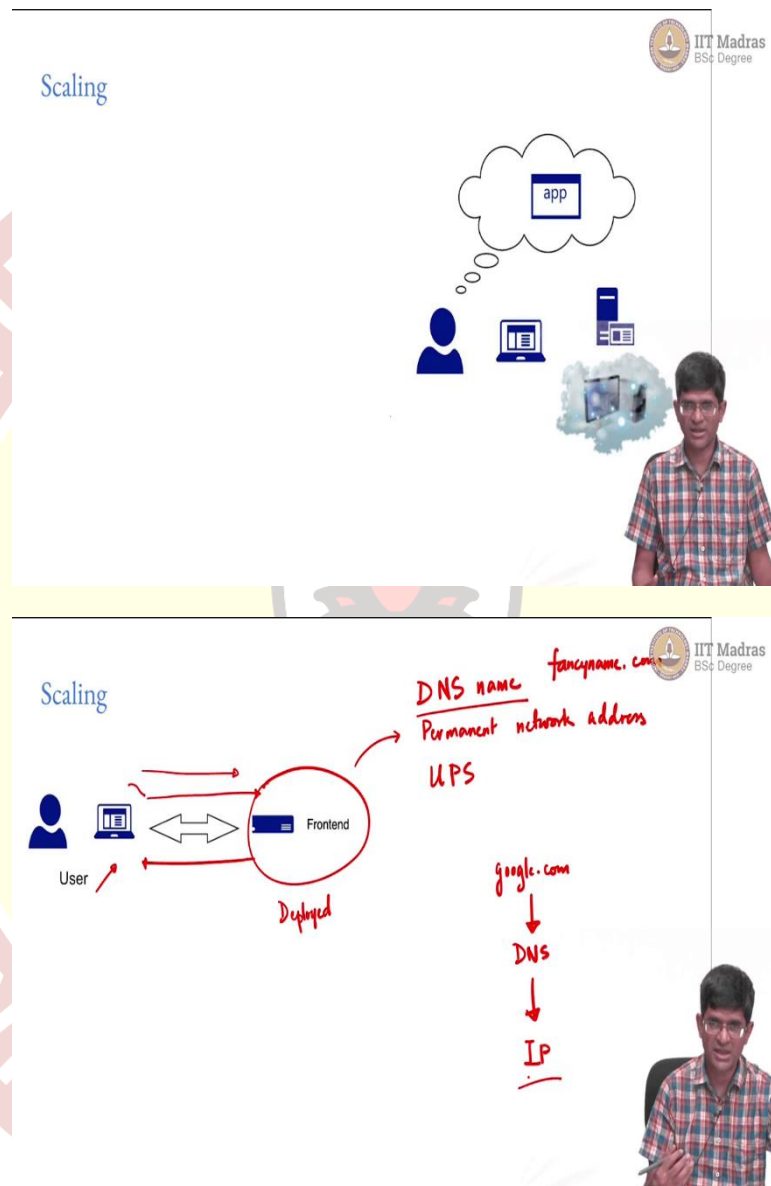
You need space, you need cooling, you need uninterrupted power, you need a network connection. So, all of that is a problem and it makes sense for some people to sort of integrate all of these things, maybe take orders from a large number of companies, set up a whole bunch of servers in one place, provide unified cooling and a good network connectivity to all of them, make sure that they cannot sort of step on each other. In other word, you want to provide each company at least that feeling that they have their own private information.

So, what kind of guarantees can you give that one company's computers are not going to affect someone else's work, things of that sort also need to be taken care of in the data center but once all that is done you essentially have the cloud. And the present state where we are now is that the cloud is pretty much publicly available. Anybody can sign up for a cloud



account. You can basically sign up for Google Cloud or Amazon Web Services or Microsoft Azure, those are three big ones but there are others. There are plenty of other services as well who will allow you to sign up and create accounts for yourself.

(Refer Slide Time: 8:28)



Now, part of the advantage of having this cloud based approach the next question that comes up is I have an app. I know that I can access it. I know that my friends have tried it out. It seems to work but what happens if like the online degree or NPTEL Swayam. I suddenly need to open it to 1 lakh users or 10 lakh users. At that point I suddenly find that the number of requests that is coming into my servers is really large. I am not able to handle it on a single machine. What do I do? You need to scale. You need to be able to respond to larger number of requests in an efficient manner.

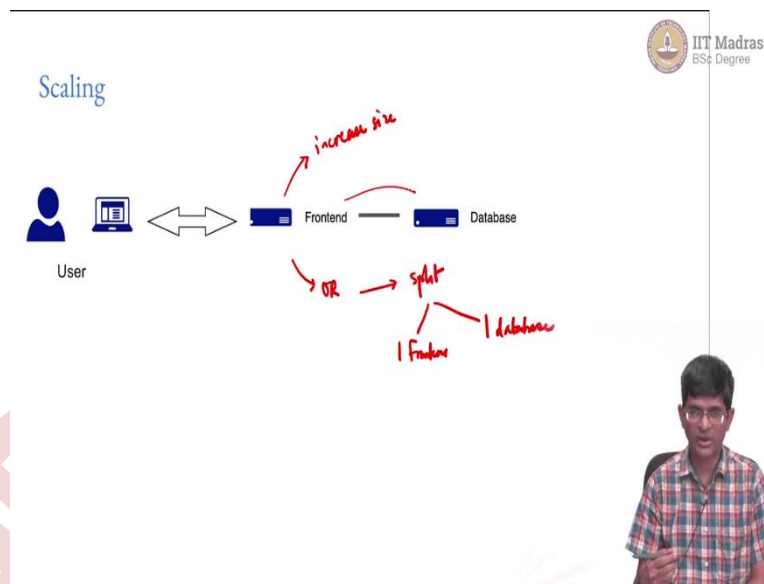
Now, of course that also requires a large part of redesigning your app itself so that it is able to scale properly but apart from that you also need some kind of infrastructure support. So, the typical use case for scaling would be something like this there is a user at their computer who is basically trying to access some kind of web service, which is provided by a frontend. So, this user is able to connect to from their laptop, from a connection to the frontend server and the server then sends back responses for everything that the user sends. All of this presumably since we are talking about a web app is happening over the HTTP protocol.

So, far so good, perfect. I mean whatever you developed on your local machine is what is being deployed on to this frontend out here. And this has a permanent network address. It has uninterrupted power and along with the network address you also probably need to have a DNS name, that is a domain name. Something whatever fancyname.com or I do not even know that there is such a website. So, please do not go and click on this link without checking it first. But you might want to sign up for whatever the name of your website that you would like it to be. So, for example google.com. Ultimately what do they do? They register with a domain name system, DNS and they have it set up such that then someone looks up for google.com. It goes through the DNS system and comes out with an IP address one or more IP addresses. And what happens is that how is this being resolved that is out of the scope of this course but it is part of the basic information and networking that we had indicated that you might need to know a little bit about at the beginning.

So, ultimately what happens is that once you have this IP address, the client's browser knows where to send the network request. It basically connects using the IP and the server knows where the connection came from, it knows where to send back the response. So, far so good. So, this is just a single person connecting to a single serve, no problems there.



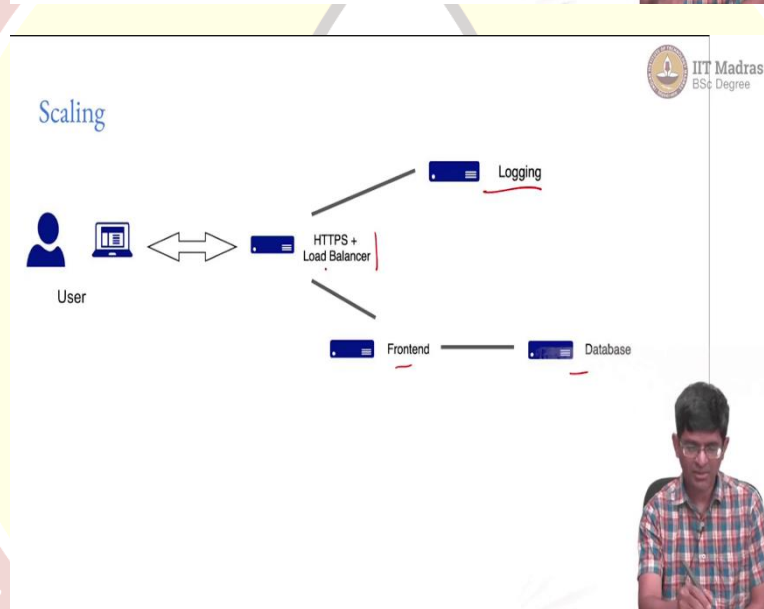
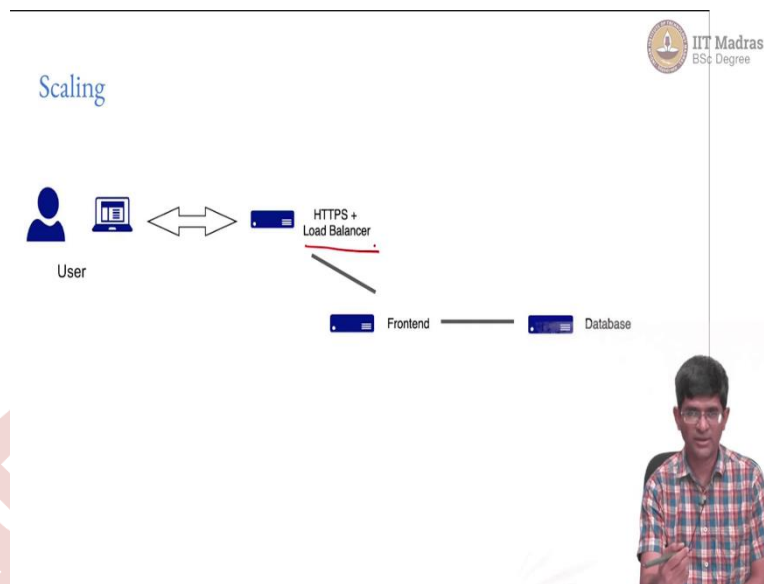
(Refer Slide Time: 11:45)



Now, the server itself, you might want to expand it. Why would you want to do that? You might find that maybe the load on the server was becoming too high and you create another machine, which handles database queries. Because as the service size increases, now you have 2 options either start making this bigger, increase size or split and make one frontend, one database. And by splitting it in this way, what you are saying is that look the database by itself requires a certain amount of RAM. It also requires some hard disk space for storage.

Now, the frontend on the other hand might have slightly different requirements. It does not require a whole lot of hard disk because it is not storing data. On the other hand it is getting a lot of network connectivity which means it has to have a very bandwidth network connection. It also might have to have something whereby the kernel is tuned to be able to accept a large number of incoming requests. But the connection between the frontend and the database is just happening over a private network. It is not happening directly from the outside world. So, I can tune that and get that to be optimized in such way that the frontend to backend or database connection happen separately. So, this is the most trivial form of scaling. You have basically split your application into 2 parts. But what happens as it continues to scale, the number of requests keep on increasing.

(Refer Slide Time: 13:28)



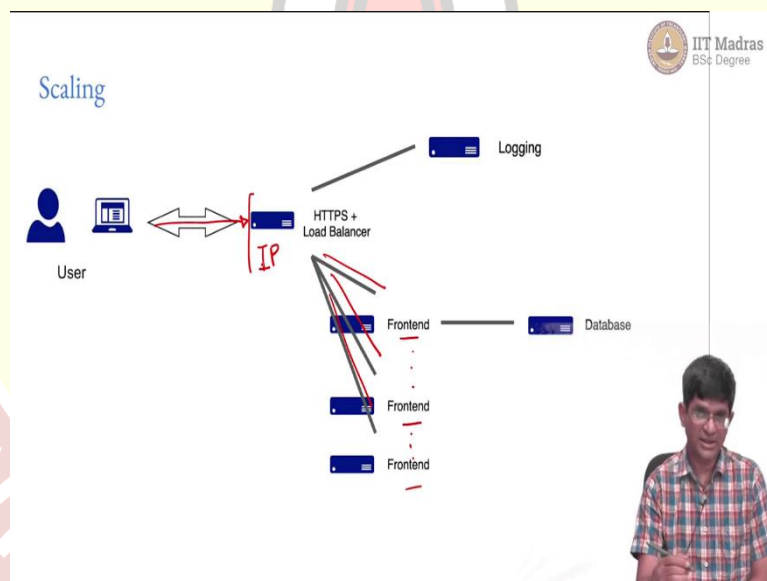
At some point, you just realize that maybe you need to put in security. People say that look your frontend is not really secure for the simple reason that you are not running on HTTPS. You need a certificate. You need a to make sure that only encrypted connections are formed with a browser. Now, either you could choose to add that on your same frontend or you might decide look let me have another machine whose only job is to handle HTTPS.

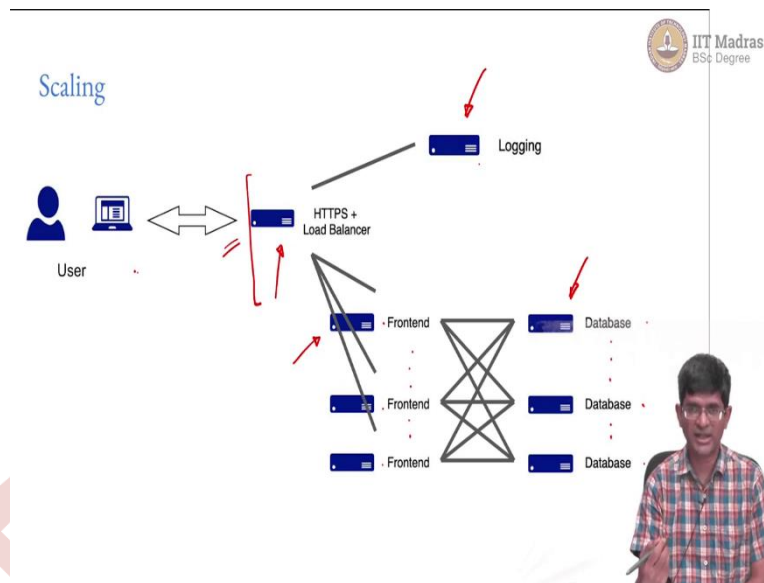
The good thing about it is if it finds that there are lot of badly found requests or something of that sort they might get dropped right at that actual frontend, the HTTPS machine. Now, that machine could also add to something called a load balancer and what that means is we will see next. A load balancer basically kicks in when you potentially have multiple frontends maybe one frontend is busy preparing the response for one user.

But for whatever reason your application is such that that takes a lot of time. So, rather than sort of trying to finish it fast or waiting until it finishes, you have some other multiple frontends that can actually respond, in which case sorry before we get to that, one of the things that you could do is this HTTPS and load balancer this frontend part of it that we have created, also needs to do one more thing which is basically storing information which you need to maybe find out for debugging later or for just checking if somebody was trying to hack into the system, were there any security related issues.

So, logging like was mentioned in one of the earlier videos is an important part of how you create an app because it helps with your debugging but it also helps with understanding where the bottlenecks are and what needs to be improved. So, rather than worrying about logging directly from the frontend or the database you could have a separate machine whose work is only to handle log-in.

(Refer Slide Time: 15:25)





But coming back to our load balancer, what the load balancer could do is let us say that I had multiple frontends. Each of these is just a machine with a different amount of RAM. I mean they could all have the same amount of RAM is just that there are multiple machines. What this load balancer will do is, it will send the first request perhaps to the first machine maybe the second request goes here, the third request goes to the third machine and so on.

On the other hand, if it finds that the first machine itself finishes its work very fast then maybe you did not really need so many different machines. You could have just sent them all to the same machine. The good thing is the user always connects only to the load balancer. So, only this load balancer's IP needs to be given to the outside world. Internally I can have multiple frontend servers or I can reduce them to one.

So, all of this whether I am increasing the number of servers decreasing the number of servers, changing where they are located, moving them from one place to another, shutting down one machine and starting another machine, all of that is hidden from the user because they connect only with this load balancer.

And of course, what you could do then further is because everything is now hidden away behind this load balancer and the user never gets to directly interact with any of this side. You could also potentially have multiple database servers. And at that point what you would say is there are multiple frontends talking to multiple database servers.

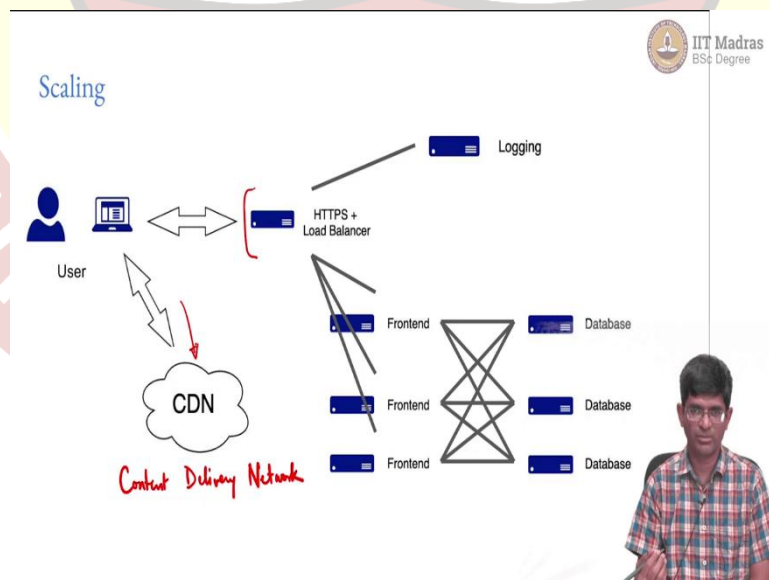
Again, everything is hidden away from this point onwards. The user only gets to see the HTTPS load balancer. They do not see what is, what kind of machine is used for logging.

They do not see what kind of machines are used on the frontend. They definitely do not get to interact with the backed directly but you might have a whole lot of machines sitting somewhere behind in the cloud that are actually responsible for coming up with a answer to any request that you sent out.

So, this is essentially how scaling gets implemented. And the thing is when you have a cloud-based infrastructure, what it means is that the cloud provider will essentially give you certain services. They will provide you with a load balancer. They will provide you with some mechanism for implementing frontends. They will provide you with mechanism for implementing backends. They will provide you with mechanisms for logging. And your app then needs to sort of put all of those things together and come up with one coherent picture which as far as the outside world is concerned just looks like a single server.

So, the user who is interacting with it has no knowledge of how the things got implemented at the backend. The cloud provider allows you the freedom to scale or to increase decrease various other modifications with what kind of machines you are using at the backend and everything can pretty much happen transparently of course provided your app also is able to support that kind of transparent modification

(Refer Slide Time: 18:28)



Now, in addition to all of this there is one more part which is often plays an important role in scaling and this is something called a CDN which basically stands for content delivery network. So, what does the content delivery network do? It sort of specializes in handing out files that are static meaning that they do not change with time and that are fairly common.

Now, what do we mean by that? An example might be you have probably heard of the term jQuery which is a JavaScript library used for various kinds of interaction. There is also the Bootstrap CSS, which is used for even in the course, we have used it for as a recommended form of CSS for styling various documents.

Now, Bootstrap there is another thing called Tailwind. There is jQuery. There is the Angular, React, all of those libraries JavaScript and CSS libraries plus certain kind of images. All of these are common. They are static. They do not really change often unless there is a new release, nothing is going to change and they are used by multiple websites. So, rather than having your web server handling the delivery of each and everyone of these files, it makes sense to in your app, in the HTML of your app, you can give a link directly to something which will download certain files from a CDN.

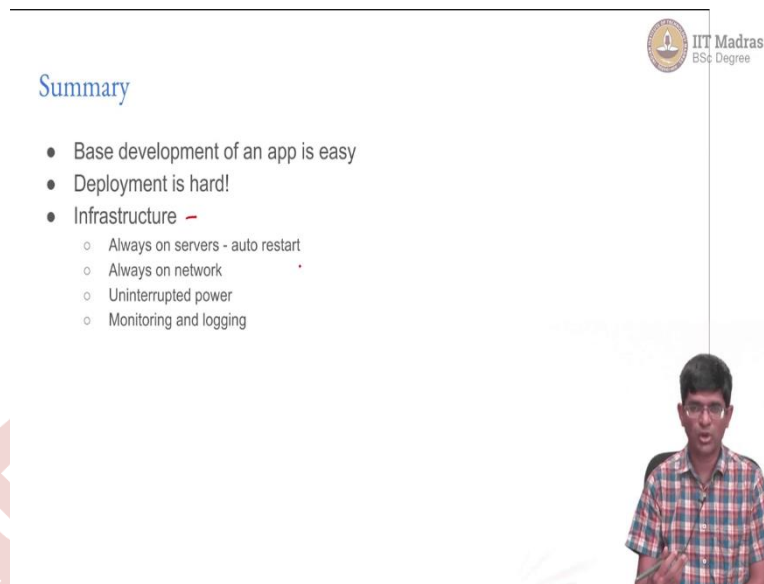
So, if you want to use Bootstrap on your website for example rather than downloading the Bootstrap CSS putting it on your website and then delivering it along with your app, it is better if you directly give a link saying download from here, download Bootstrap from here. Now, that is good for 2 reasons. One is it reduces the load on your server. The second is because Bootstrap is used on a large number of websites. There is a good chance that it was already cached and present in your browser even before you got to this point.

So, ultimately, in other words this should hopefully give you a picture of what the final deployment of an app looks like. It is not just a question of writing a piece of code and putting it on a machine, you need to also think about where you are going to deploy it? What kind of frontend do you need? What kind of content delivery network do you need? Whether it is already set up and you can make you use of it? Do you need logging? Do you need multiple machines frontends? Do you need multiple backends? How are they going to interact with each other? What kind of cloud service provider gives you all that functionality? A whole lot of questions that need to be resolved in order to actually come up with a proper deployment.

Now, the details of that are out of scope of this course. This is more to sort of give you an idea of what is involved and to sort of get a picture for what are the things you should be looking at if you are really interested in moving forward with these topics.



(Refer Slide Time: 21:26)



Summary

- Base development of an app is easy
- Deployment is hard!
- Infrastructure
  - Always on servers - auto restart
  - Always on network
  - Uninterrupted power
  - Monitoring and logging

So, to summarize, the base development of an app is relatively easy. We have been through developing an app using flask and I mean if it is a simple app, the development is easy. All that you need to know is python and some amount of HTML, CSS, all that is depending on how much you want to style the app if it is just some basic text and other thing you do not even need to worry too much about that.

But deployment can be quite hard especially if you want to scale. If you are only creating an app that that only you are going to use or maybe five of your friends are going to use then you do not need to worry about any of this, run everything on a single machine. It is quite possible to rent a machine somewhere on the cloud for fairly low amounts these days, maybe 5 dollars a month, yeah it is not, it is debatable whether that is really cheap but it is definitely not expensive. 5 dollars a month is probably similar to what you are paying for your cell phone bills

So, if that is the case then you could have your own server somewhere in the cloud and based on that server you could use that in order to run quite a lot of applications of your own. The problem is it is not going to scale well if large number of people try connecting, which is where all the problems come in.

So, there are providers on the other hand who give you that infrastructure and what is the kind of infrastructure you need? You need always on servers especially things that will automatically restart in case there is a crash in the system, is the crash because of the software bug or is it because of power outage, does not matter the server should ensure that it restarts.

The network should be always on and high bandwidth. There should be uninterrupted power. There should be monitoring to see that everything is okay. There are no problems and also logging what are all the events. Events could be related to your app or it could be related to the machine. So, all of this is essentially what is provided by so called cloud service providers and how do they go about doing that, what are different options that you have over there that is something that you need to look further into when you are looking at deploying an app.

