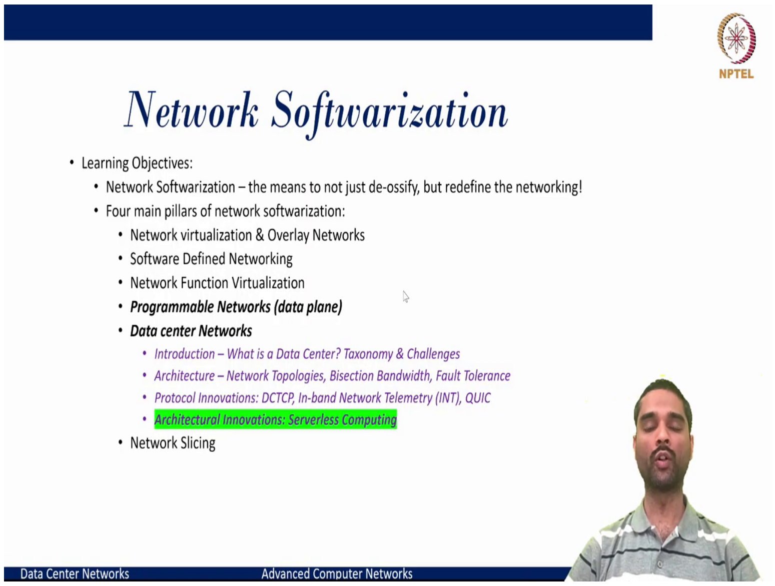
Advanced Computer Networks
Professor Doctor Sameer Kulkarni
Department of Computer Science Engineering
Indian Institute of Technology, Gandhinagar
Lecture 56
Serverless Computing - Part 1

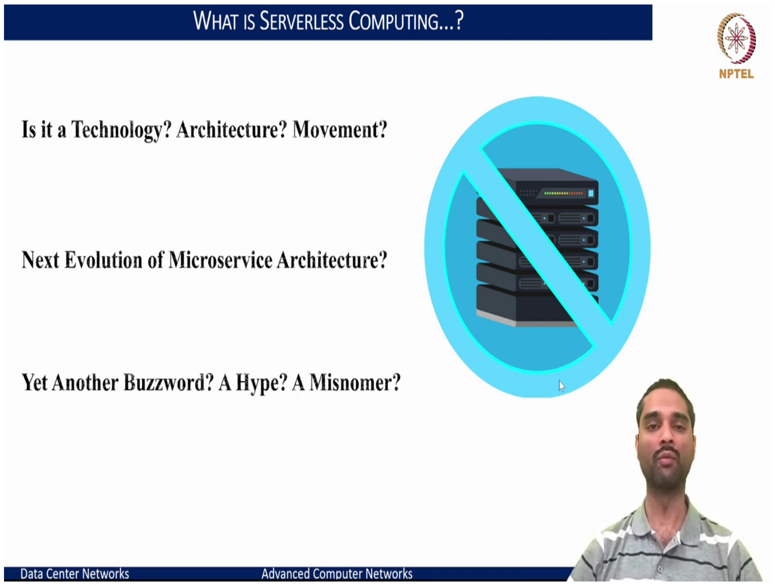
(Refer Slide Time: 0:17)



Network Softwarization

- Learning Objectives:
  - Network Softwarization – the means to not just de-ossify, but redefine the networking!
  - Four main pillars of network softwarization:
    - Network virtualization & Overlay Networks
    - Software Defined Networking
    - Network Function Virtualization
    - ***Programmable Networks (data plane)***
    - ***Data center Networks***
      - *Introduction – What is a Data Center? Taxonomy & Challenges*
      - *Architecture – Network Topologies, Bisection Bandwidth, Fault Tolerance*
      - *Protocol Innovations: DCTCP, In-band Network Telemetry (INT), QUIC*
      - *Architectural Innovations: Serverless Computing*
    - Network Slicing

Data Center Networks Advanced Computer Networks



WHAT IS SERVERLESS COMPUTING...?

Is it a Technology? Architecture? Movement?

Next Evolution of Microservice Architecture?

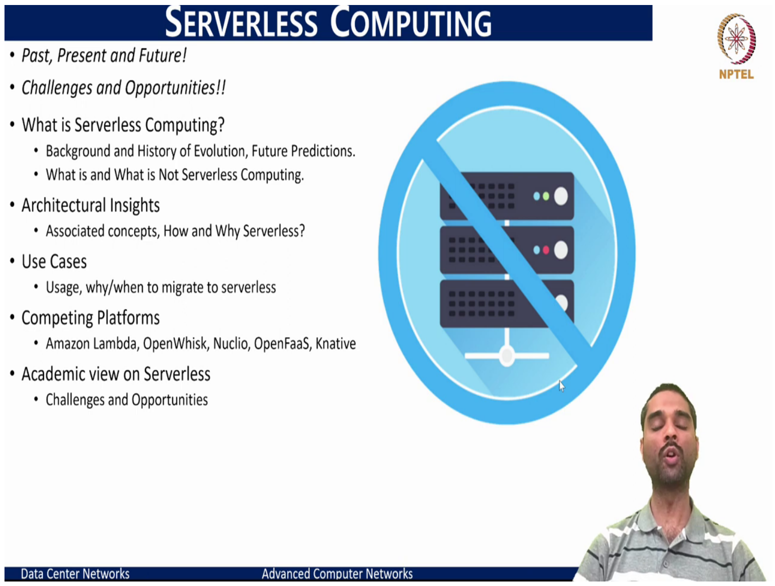
Yet Another Buzzword? A Hype? A Misnomer?

Data Center Networks Advanced Computer Networks

In this lecture, let us try to cover some of the glimpses of the architectural innovations that came in the rights of data center networks with the new paradigms of computing what we call as Serverless Computing. Perhaps, you might have already heard about serverless as it is been a hype over the last few years or even worked to some extent in serverless.

So, let us try to learn what Serverless Computing really means. So, is it a technology, an architecture, or a movement? Like, how do we really define serverless computing? And to put it in another context, is it an evolution of microservices architecture that has been prevalent over the last few years or just a hype or a buzzword that is creeping in, or is it completely a misnomer? So, let us try to get some aspects on each of these in terms of where serverless computing really fits and what exactly it is.

(Refer Slide Time: 1:08)



SERVERLESS COMPUTING

- *Past, Present and Future!*
- *Challenges and Opportunities!!*
- What is Serverless Computing?
  - Background and History of Evolution, Future Predictions.
  - What is and What is Not Serverless Computing.
- Architectural Insights
  - Associated concepts, How and Why Serverless?
- Use Cases
  - Usage, why/when to migrate to serverless
- Competing Platforms
  - Amazon Lambda, OpenWhisk, Nuclio, OpenFaaS, Knative
- Academic view on Serverless
  - Challenges and Opportunities

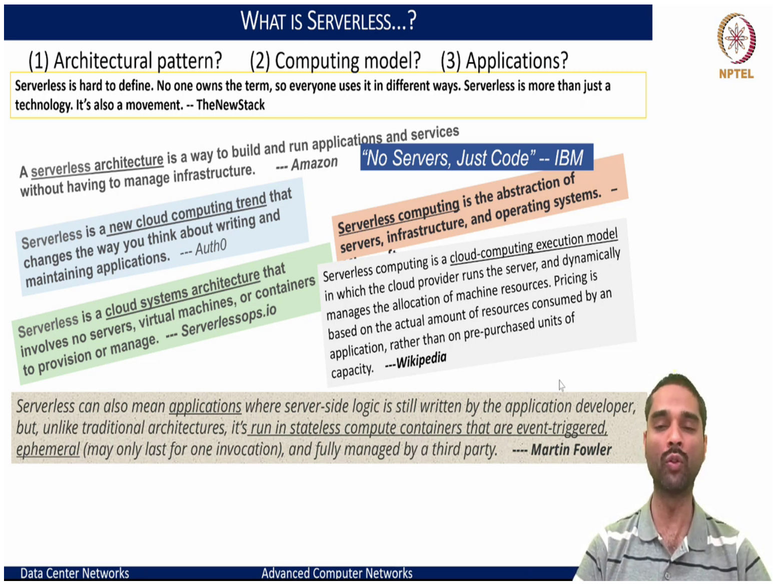
Data Center Networks Advanced Computer Networks

And to do this, let us try to cover a bit of background and history of the evolution in terms of how it all started and what else in the future in the context of serverless computing to some extent and try to understand and get some insights about the key concepts associated with serverless computing and why it really matters.

And what are the key benefits and advantages that it brings to the table? And try to put in the perspective of use cases to understand what really fits in the serverless computing model and what does not and what kind of use cases we can think of that would really be useful in this context. And there have been so many of the variants of platforms that have been facilitated over the cloud, like Amazon Lambda, OpenWhisk, OpenFaaS, Knative, and so on.

Even Google Cloud Functions, Microsoft Azure functions, to name there are quite a lot. And it may be worth also to look if time permits into these aspects, and the most critical aspect would be also to look from the academic view in terms of what challenges and opportunities lie in serverless computing and see how we can make some contributions in this space.

(Refer Slide Time: 2:18)



WHAT IS SERVERLESS...?

(1) Architectural pattern? (2) Computing model? (3) Applications?

Serverless is hard to define. No one owns the term, so everyone uses it in different ways. Serverless is more than just a technology. It's also a movement. -- TheNewStack

A serverless architecture is a way to build and run applications and services without having to manage infrastructure. --- Amazon

"No Servers, Just Code" -- IBM

Serverless is a new cloud computing trend that changes the way you think about writing and maintaining applications. --- Auth0

Serverless computing is the abstraction of servers, infrastructure, and operating systems. –

Serverless is a cloud systems architecture that involves no servers, virtual machines, or containers to provision or manage. --- Serverlessops.io

Serverless computing is a cloud-computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity. ---Wikipedia

Serverless can also mean applications where server-side logic is still written by the application developer, but, unlike traditional architectures, it's run in stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a third party. **---- Martin Fowler**

So, let us start with the basic question of what is serverless? And many people have tried to define it in many other ways, in many different ways as well. So, the question was how can we consolidate and build one, what are the pictures that different definitions are painting? And as it was put up by TheNewStack, they said it is very hard to define and no one owns the term serverless and everyone uses it to their benefit in their own context in different ways.

And to sum it up, it is basically more than a technology, or rather it is a movement, and that is how TheNewStack put it. But as computer engineers let us try to see if is it an architectural pattern and many tried to define a serverless as an architecture to build and run applications like what Amazon when it started Amazon Lambda, called it a new architectural model where you could run applications and services without having to manage infrastructure, and this was the definition from Amazon. And likewise, the Serverlessops.io defined this as a cloud systems architecture in specific to say that you are trying to build architecture that involves no servers, VMS, or containers that you need to provision or manage.

Again like if you see the context both try to mean in terms of what machines or what aspects you would manage on the cloud and in that essence the serverless architecture is to say that you just manage or build your applications and manage none of the infrastructure.

The second way to think of it is, is it a computing model? Because we have been saying serverless computing. And there were also definitions that tried to put in this aspect like when the definitions came as a serverless computing to be an abstraction of servers, infrastructures, and operating systems and this was put forth by Microsoft to say that this is a computing model. And likewise, if you look up Wikipedia, it says it is a cloud computing execution model in which the cloud providers run the servers and dynamically manage the allocation of the underlying resources.

And likewise, they manage and control the pricing from the user's point of view, and this is where the serverless stands as a computing framework in which users who want to do some compute can go, offload the computation to the cloud, where the user did not have to worry about the infrastructure, the operating system or the server, the hardware themselves. So, in that essence, this fits as a computation or a computing model.

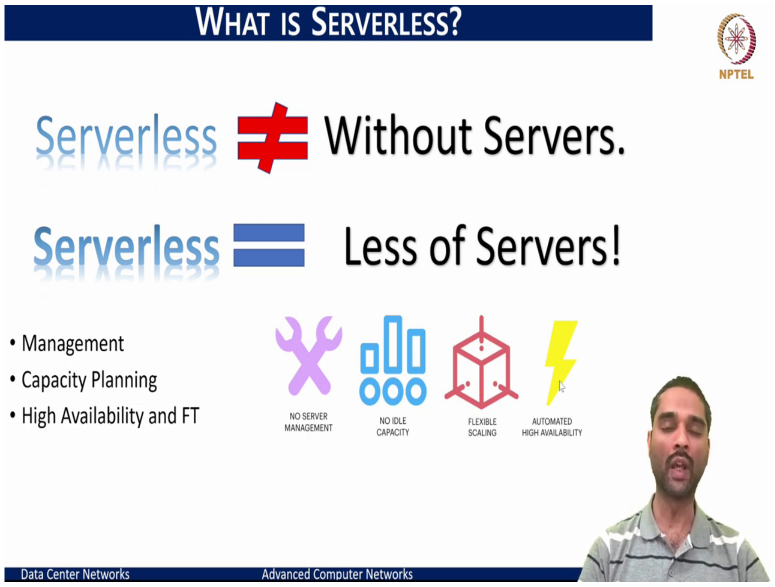
And third, is it just then that we are speaking of building the applications because we have been already familiar with writing codes, and this is what IBM tried to put, saying, ‘No servers, but just code, and this is what every developer is a lot more familiar with than looking into the nuts and bolts of the hardware and try to adapt the system. And serverless tries to bring this context of what we say as applications where users are only going to write code.

And the way it was put by Martin Fowler is serverless can also mean applications that means whatever is the server-side logic that needs to be written, you write it as an application developer without taking into aspect any of the traditional architectural paradigms. Just think of writing a function that will be called whenever and wherever necessary. In essence, then it becomes like a very small unit that is going to be triggered and executed and lasts just as much as the time that it needs to execute.

And this is where the application side of development you can send this in the serverless. So, to sum it up all serverless is as well architectural pattern in terms of how you develop and manage your applications. It is a computation model in terms of how you want to build your computational logic by abstracting many of the nitty gritties of hardware and software, especially the system software and the hardware abstractions.

And it is as well the applications where the developers are now focused just on writing the code that they need to, without worrying about when it will be called, how it will be invoked, what is the server of which physical location it is going to run and all of these aspects.

(Refer Slide Time: 6:28)



What is Serverless?

Serverless ≠ Without Servers.

Serverless = Less of Servers!

- Management
- Capacity Planning
- High Availability and FT

NO SERVER MANAGEMENT
NO IDLE CAPACITY
FLEXIBLE SCALING
AUTOMATED HIGH AVAILABILITY
NPTEL
Data Center Networks
Advanced Computer Networks

So, what we want to say is that serverless does not mean that it is without servers, you still need your code to be run on some compute node and when we are talking about the clouds, which are basically the constituents that we learned about the data centers in the earlier lectures, you would have to run them on one of the servers that is mounted in one of those racks.

And that means it is definitely not that the code just lives without servers, but the code that executes on the servers but offloads you from understanding any of the nitty gritties of those servers. So, serverless in essence, means less of servers or less to do with the understanding of the physical aspects or the constraints of the servers.

And what this means, like if you have a VM, then if you compare it with that model, you have to manage the VM in terms of how many CPUs you would have to allocate, and which physical machine you would want to run that. So, there are a lot of management aspects that go when you want to deploy those as VMs or as physical hardware on which you want to run the code.

And serverless completely takes that aspect off to say that no more server management; you just write the code, and this can be run on any type of hardware that may seem fit to run it, and that can be completely decoupled now from the one who wants to own that code or develop that code.

The second is capacity planning, and this is where most of what we spoke about data centers matters a lot. When we want to do anything at a cloud scale, we need to understand, like when we build a system, what is the bandwidth in terms of network connectivity that it can offer, what the CPU it needs so that it is able to scale to facilitate so many users, what is the memory requirement for this VM that you would want to provision or a container that would want to set so that it can support so many of the users.

So, scalability has to be pre-planned when you deploy those. But now with serverless, it says you do not need to worry as much about capacity planning, and what serverless provides is to say that you can have as much of the scale of instances that you would want to bring. So, that is where the flexibility comes in terms of scaling whatever the code that you would want.

And also like if we do provisioning for the worst case we would end up saying that there are a lot of idle capacities, especially this is true with networks and more servers that we see, you barely have more than 50% utilization of any of the devices that would be running. So, that is basically a wastage and utter utilization of resources.

So, serverless even takes out that aspect to say that you would run, you would use those only when you need it, and if you do not need any hardware at that point if I am only executing my code I need hardware on which it can execute, but if I am not executing anything it does not have to be glued to the hardware to say that I have resources that are reserved for this function and that is the benefit in terms of capacity planning this serverless computing brings.

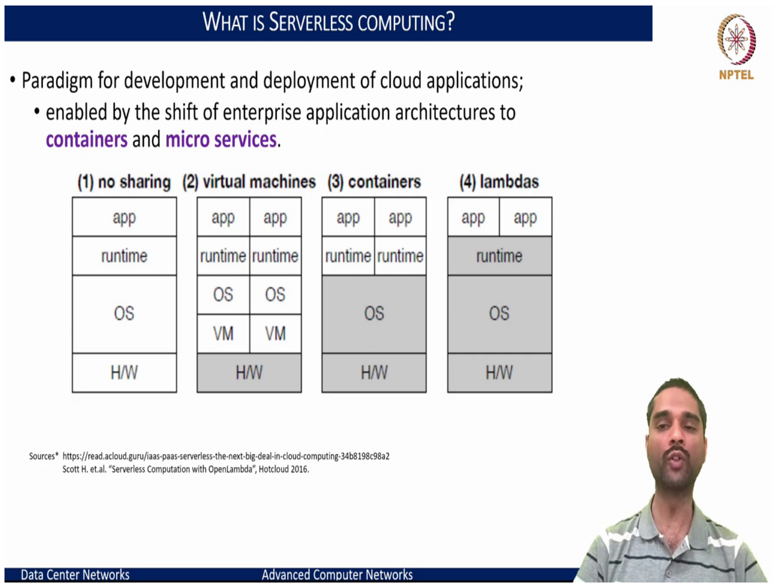
And another most important critical aspect that it brings to the table is about how we manage your systems for high availability and fault tolerance. We spoke about earlier when we discussed about the NFV and said, “Telecom grade requires you to have 99.999 or the five 9 availability with less than a few seconds of downtime.”

And this meant that we had to plan for the failures, we had to plan for the updates that we had to do when we had the VMs or containers or even the hardware on which we wanted to run our services. Now, serverless says, “We can even take that abstraction, provide the abstractions for that model to say that we can automate the high availability and ensure fault tolerance in the best possible way.”

What this means is you are no more going to worry about having to run multiple of the instances' primary, secondary or active or active backup, but run these instances as and when necessary on any of the hardware. So, now we have in essence, decoupled the functions from the hardware just as what we spoke in NFV and we have decoupled the management aspect of these from the hardware as well.

And we can run these functions on any of the hardware that is available at a given point in time. It is only about migrating the code towards the hardware on which we would want to run these functions and that is how serverless provides these things with respect to making the developer or the deployer's job to not deal with the management, capacity planning, or HA and FT aspects.

(Refer Slide Time: 10:59)



WHAT IS SERVERLESS COMPUTING?

- Paradigm for development and deployment of cloud applications;
  - enabled by the shift of enterprise application architectures to **containers** and **micro services**.

Sources* https://read.acloud.guru/iaas-paas-serverless-the-next-big-deal-in-cloud-computing-34b8198c98a2
Scott H. et.al. “Serverless Computation with OpenLambda”, Hotcloud 2016.

Data Center Networks Advanced Computer Networks

So, to put it formally in terms of how serverless computing can really be defined, as I said, it is just a paradigm for the development and deployment of cloud applications, and when we say cloud applications, typically, these are the network applications that we want to build, not necessarily that you build them on the third party clouds. We could even build these on the on-premise clouds as well.

So, wherever we see that there is an offload of computation and there are specific characteristics of computation that need to be done outside of our machines we can think of trying to bring this serverless computing paradigm as much as possible if applicable. And what this enables us is to shift from the enterprise-based application architectures that we typically see, that is, monolithic architectures which were very prevalent in the early 90s, which switched to two-tier, three-tier architectures and then gradually towards the containers and microservices and serverless bits on top of this container and microservice-based functionalities.

And what I am trying to show in this diagram here is to say that when we had just the physical machines, we would own completely the hardware, completely what OS we would want to install on it, what is the necessary runtime, the libraries that would have to patch, set the things up and then build the applications and run the applications on top of this segment.

An application would be running on this dedicated hardware and software and essentially, all of the management needs to be done in terms of what goes underneath the stack in each of the layers, and all of these are the things that we need to care. So, what are all the items marked here in white are essentially the aspects or points that we should consider and care about when we are trying to deploy such applications.

And this also means that there is no sharing of resources whatsoever in terms of hardware or software if it is a tightly coupled app that is going to run on its dedicated hardware and software. And we moved towards the virtual machine model where we would have a single physical hardware which would then run multiple instances of the virtual machine.

So, we virtualized the hardware, and we would run a software stack on top of these hypervisors where we would have the operating system, the runtime, and applications that would run on top of VM, and they are all going to be sharing the underlying same hardware. So, we brought in some notion of the sharing of the resources.

At the same time, we also maintained a nice partition or isolation from the application point of view in terms of if things fail, it is going to work, and have no impact whatsoever on the other entity. And this was the lateral movement that we saw which gained attraction for the cloud computing and data center workloads that started to evolve.

And gradually we also shifted towards what we call as a OS-level virtualization where the hardware and operating systems would now be shared while the applications would have their own run times, and that is the limit of isolation that each application would be confined to. And this, even if we compare this model with the virtual machines, we see that there is limited isolation as opposed to what you will see with respect to the VMs.

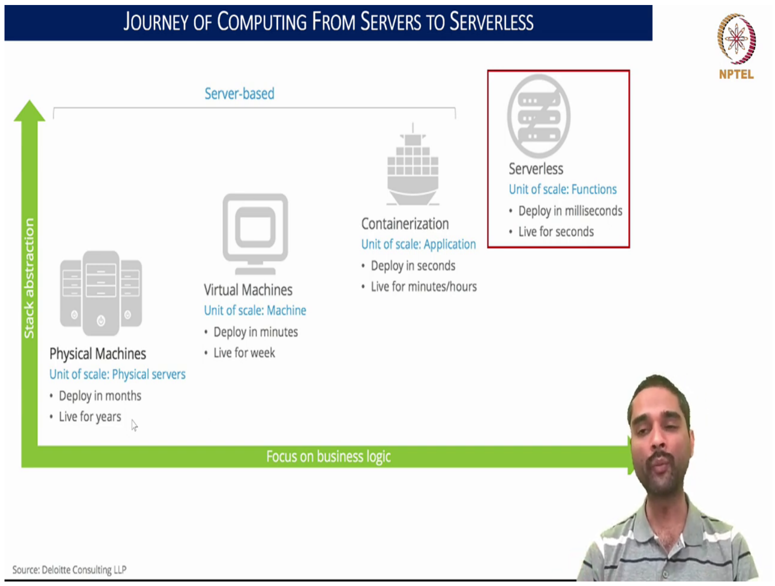
But there is still an isolation from the point of view of an application in terms of what run times it would want to support and how it would want to build. And what this also means is we are going to share the same hardware and operating systems without having to re-build the different operating systems for each of the VMs. So, there is much more sharing that is done in this case.

And there is much more of also management in terms of both the hardware and operating systems that are necessary to facilitate the containers is being done, but it alleviates the job of trying to say I have to explicitly manage the OS for each VM. So, you are going to share these resources and less of a burden from the developer's point of view in terms of managing these aspects.

And gradually now in serverless as an example here, the Lambda services from Amazon, what it really matters now is to say that none of the hardware, operating system or runtimes is the concern of a developer. All of these are going to be like the shared resources that you are going to be using and a cloud service provider would manage all of these entities.

And as a developer, you would only be working and looking for the applications that you would want to build and this is what really as a developer or a service entity we would really be caring about. And this is the flexibility of this serverless paradigm where the entire scope of putting things is just confined to building the right applications.

(Refer Slide Time: 15:43)


JOURNEY OF COMPUTING FROM SERVERS TO SERVERLESS

Server-based

Physical Machines
Unit of scale: Physical servers
- Deploy in months
- Live for years

Virtual Machines
Unit of scale: Machine
- Deploy in minutes
- Live for week

Containerization
Unit of scale: Application
- Deploy in seconds
- Live for minutes/hours

Serverless
Unit of scale: Functions
- Deploy in milliseconds
- Live for seconds

Stack abstraction

Focus on business logic

Source: Deloitte Consulting LLP

So, if we see our journey from the computing point of view, how we have evolved, we started with the physical machines where the unit of scale would be just a physical server that you would want to replicate, but to deploy them, it always took months because you have to procure the resources, hardware, get them and then plump them.

And once you set it up the network, connect the things, and then power on they would live for years, and they would continue to live when you would modify whatever applications that you would, whatever the OS that you would have to upgrade any software that you would want to patch, all of them on this machines.

Second, we transitioned from these physical machines to virtual machines because of the hardening that we see in terms of managing these devices, in terms of the timeline they need for us to deploy any aspects, or even in terms of when we say about utilization, they are just plot and plumped, they may see sometimes good utilization and many a times underutilization.

So, if you have to patch, we could reuse the same physical machine to facilitate multiple of the virtual machines, and this also brings a lot of cost benefits in terms of reusing the same capital expenses to facilitate multiple VMs and also in terms of operation, now it is softwareized, it makes it a lot more easier.

And with virtual machines now we can control and manage how to deploy with just one click and maybe it takes a minutes, few minutes to boot, but it is still much more easier than to say I go to a particular location power on, or power off the machines. And this was the benefit that was leveraged heavily by the cloud and data center networks that we saw where you would have most of the workloads being run on virtual machines.

And then the virtualization continued and we moved towards the OS level virtualization which we call as still lightweight containers and this allows a lot more flexibility than VMs in terms of making them lightweight because you do not have to contain the operating system within a VM to run the VM. And also now it is just the application center and times that you really care about for what applications, what kind of runtimes you would need, and then deploy them.

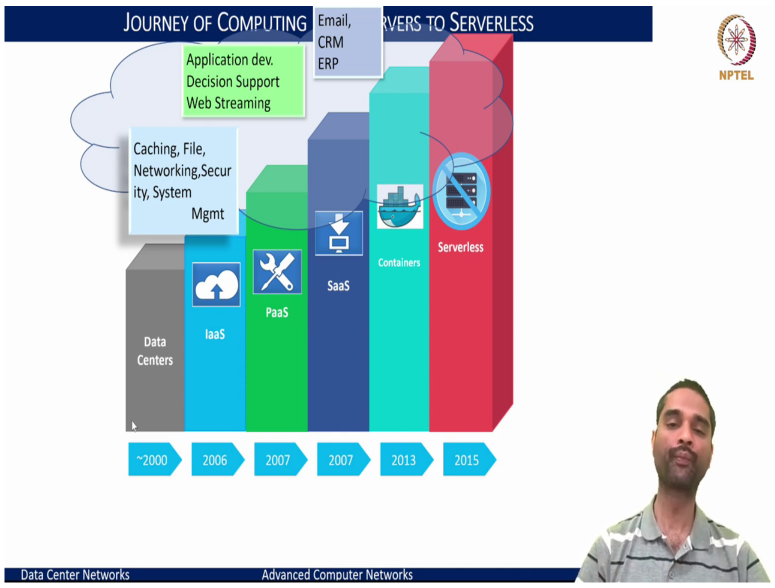
So, the deployment times are also very short in terms of very few seconds. And then you could run those containers whenever necessary and make them work as long as you would need and then take them off if you do not. And this model was all what we call as a server-based model in terms of how the networks or how the computation stacks have been used and what were the key focus on the business logic in terms of how you would deploy and run.

Basically manage and ensure that these services can be provided to the end user through the use of these server-based models. And going forward what we see is the serverless architecture where you would just care about the functions as the key units of scales that you would want to build.

And the architecture that we are going to learn about is serverless, which now makes the unit of scale as a function that you would want to deploy and brings down the deployment time drastically to sync in a few milliseconds and even less than a millisecond you could spawn several of these serverless functions.

And the other important aspect is you would want to run these functions only when you need them. That is if there is a need to do some processing I would run that function, if there is no need I would not need any of the compute cycles or processor power to make and run these code. And that is why these become ephemeral in one context showing they are very very short-lived and live for a very few seconds as the need be.

(Refer Slide Time: 19:06)


JOURNEY OF COMPUTING ... RVERS TO SERVERLESS
Email, CRM ERP
Application dev. Decision Support Web Streaming
Caching, File, Networking,Security, System Mgmt
Data Centers
IaaS
PaaS
SaaS
Containers
Serverless
~2000
2006
2007
2007
2013
2015
NPTEL
Data Center Networks
Advanced Computer Networks

So, to put in the perspective of the data centers that we learned and this wall emerged in the early 2000s and then we moved to the cloud with the very core concept being infrastructure as a service. Andwhenever Amazon, any cloud service provider would provide an infrastructure the one who would rent, one would be a tenant would have to manage all of the aspects of how to build the networking, how to build the security framework around them, what is the resources, how to manage the compute and memory resources, the entire system management, what kind of caching or file system that you would want to use? all of these aspects had to be learned by the one suit rent out the infrastructure from the cloud service providers. And then we gradually moved to the platform as a service model.

And here platform which encompasses the underlying hardware infrastructure as well as the key functionality is the operating system, the runtimes that the users would want to build as a means to provide whatever the middlewares that you would want to build, it would all be readily developed by the platform. And this facilitates a framework for any application developers to use and build it.

So, that way this platform as a service model like the Google cloud that we typically use. It is a built on a Google cloud platform where this platform allows us to run and deploy any of the applications and the overall framework of kind of tools that they, the cloud service provider would facilitate.

And this is where the focus now from a developer point of view would be to build the right set of applications and trying to build the key decision support around these applications when they are going to be deployed in those platforms, and this could involve like streaming or web applications that you would want to build.

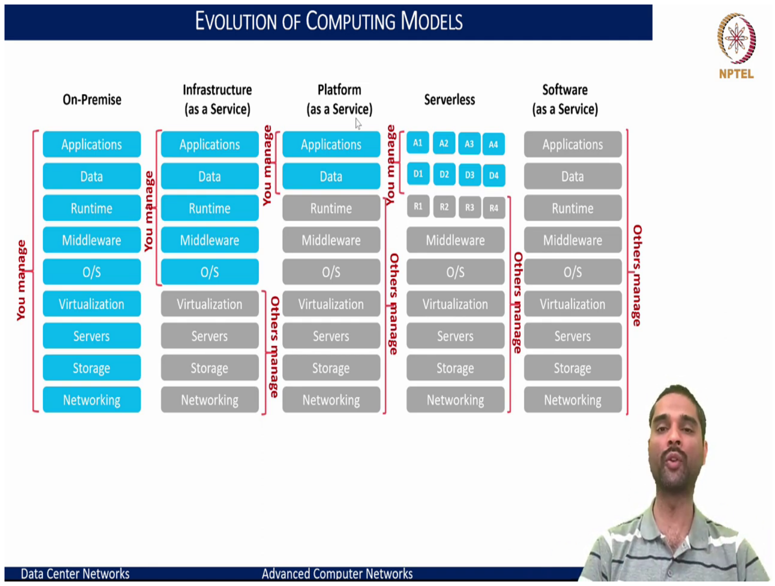
And gradually, we moved to also like around the same time; in fact, when IaaS started and PaaS came, the software as a service model also started to come and this was very effective in the sense of trying to say that you do not have to build the applications that I want to deploy on your machine, but rather I deploy the applications on cloud and allow any user who can get access to the cloud to use them, for example, the email services. And we will just connect to the email server that is somewhere on the cloud, connect, and use whatever we are. The entire application logic is integrated, provided to us as software and this forms the basis of a service. And likewise, the CRM tools and the ERP tools that most companies use in terms of trying to manage their resources, everything is offloaded to the software that is being built end-to-end deployed in a cloud and provide the services on the host, on any of the premises that you would want to use.

And this is where Salesforce and many other companies started to build the software as a service. And all of these models, eventually, were using in some way the virtual machines as a means to scale, and then with the advent of containers, the OS level virtualization model with a lightweight containers gradually replacing the virtual machines in when and where possible.

And we saw this container revolution that started to take shape around 2013. And now we are seeing the growth of what we call as a serverless computing model where containers become the essential components that you would want to run, but also take out the aspects of ensuring that the runtimes that you would want to build or develop or the infrastructures where you would want to build or manage, all of these aspects to be taken out.

And this is where serverless computing builds on top of these IaaS, PaaS, and SaaS frameworks, leverage into containers to deliver what we called as function as a service. In fact, the entire function that can be deployed and developed as a model of software as a service can now be translated to be provided to an end user as a function as a service.

(Refer Slide Time: 23:10)



And to put this in perspective and see what really matters and how things change. If we see this on-premise setup, whenever we want to do on our, let us say our desktops or within our campus networks or in enterprise we have to define all the way from the entire stack starting from the application's point of view what we want to build and for those applications what kind of data and runtime that we need to be supporting, what is the database that we need to set up, what kind of middlewares that we would want to have, what is the operating system, whether we want to use virtualization layers, like the hypervisors that we would want to use and including the hardware at the bottom in terms of what kind of server storage and networking infrastructure that we want to build. So, this entire stack is what we need to build and manage in terms of the life cycle for providing any of the applications.

But as we move to the infrastructure as a service, we are completely abstracted out of the infrastructure management because we rely on the cloud service provider to manage all of these networking storage service, server compute, and memory requirements as and how we rent from them and also maybe transparently to the virtualization on top to facilitate as a VM that we can run on their infrastructure.

So, when we do that what we really manage then is basically within a VM what kind of an OS that I would deploy, what are the middleware runtime, you can see this is purely the software stack that we are trying to manage within an infrastructure. And then, if I want to connect multiple of the VMs, I would want to set specific rules and manage the networking components within the software stack.

But how they are really going to be networked, whether it is going to be a virtualized network or physical network, all of that would still be managed by the cloud service provider. And when we move into the platform as a service, we further see that now as a platform, the cloud service provider would take care of the operating systems, the middleware, and the runtime completely and let the developers or whoever wants to have access to the platform develop and manage just the applications and data that you would want to build.

Like if I want to build a Google function, I care about what kind of data I operate with and what kind of application I want to build and how do I store that data, access the data, within a runtime that the Google Cloud supports. And this is where the platform as a service model takes out much of the burden in terms of how we manage the hardware.

In fact, zero hardware management, but just the software that we would want to build and what are the components of the software that we want to build. By leveraging the runtime as the Google framework APIs that we want to use to manage our data and applications that we want to run.

And continuing on the same lines, now, if we want still further flexibility, we could as well break this to say that we can build the server functions without worrying about the runtime and build those as microservices rather than trying to build one big application that is trying to do many things, we could incorporate the model of microservices to build and do just one specific function at a time.

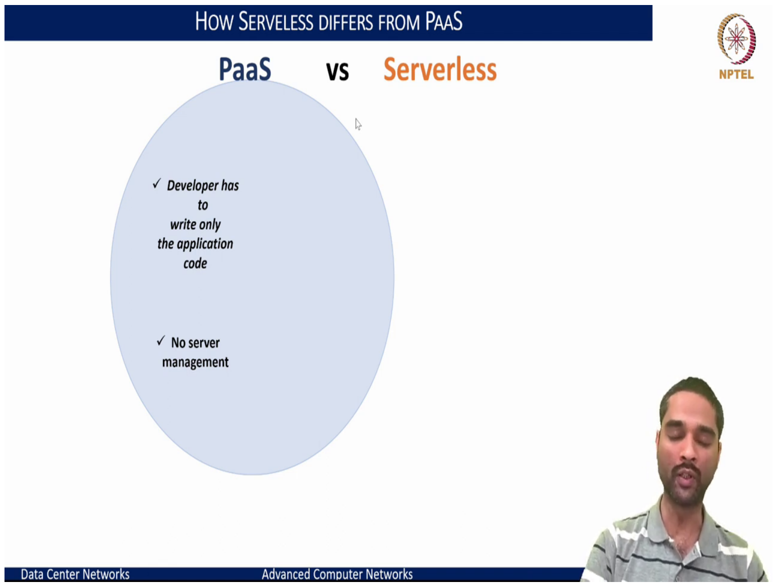
So, we can think of application being decomposed to a multiple of the sub-functions that we would want to run, and that way even the entire data can be chunked to say what data each function needs to look at. And this is a paradigm shift in terms of showing how we want to treat and see the functions to be and the associated data that they need to operate on.

And this also gives us a lot more flexibility in trying to say what kind of a runtime that we would really want to suit a particular application. So, we can have different kinds of runtimes that we can build our applications to better fit rather than just sticking to one kind of a runtime that would be used for the entire application.

And if we now juxtapose this with software as a service as an end user you really do not have any control or aspect to manage with respect to the software, the entire end-to-end software is managed by the service provider, whoever like the Salesforce or whoever, they would manage the entire thing, and there is no control or aspect that you would see when it comes to the software. You just use the software whatever is being provided on the cloud as it is. So, you can see that the serverless in a way integrates the best of the platform as a service and software as a service model to say how you have the control over and what kind of applications you want to build.

But you do not have to worry as much about the platform APIs that you want and you may want to customize in the way that you want the things to be built and ensure that you are able to provide this with the custom software that you would want and develop. So, it is good to understand in this context how the serverless really behaves and what are the distinct characteristics for a serverless with respect to the platform as a service as we see that the management aspects here look the same and understand what aspects we need to be distilling in the serverless model?

(Refer Slide Time: 28:09)

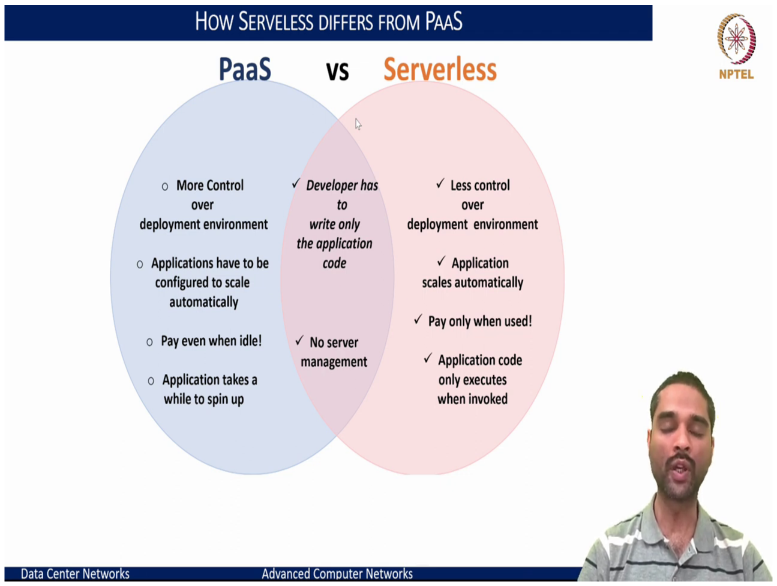


How Serveless differs from PaaS

PaaS vs Serverless

- ✓ Developer has to write only the application code
- ✓ No server management

Data Center Networks Advanced Computer Networks



HOW SERVERLESS DIFFERS FROM PAAS

PaaS vs Serverless

PaaS	Both	Serverless
○ More Control over deployment environment	✓ Developer has to write only the application code	✓ Less control over deployment environment
○ Applications have to be configured to scale automatically	✓ No server management	✓ Application scales automatically
○ Pay even when idle!		✓ Pay only when used!
○ Application takes a while to spin up		✓ Application code only executes when invoked

So, let us try to see how serverless differs from platform as a service. So, when it comes to platform as a service, developer has to write just the application code, and this you would be doing on top of the runtime that is being provided as a platform to the user using the APIs that would be allowed to use and you basically manage your code and the associated data so that it simplifies the developer's job.

And second there is no server management, neither the operating system stack nor the hardware. So, that relieves a lot of burden from the developers. And these are the benefits that platform as a service brings, and what it also does is to provide control over the deployment environment in terms of you choose the platform where you would want to deploy and govern what characteristics of the deployment environment that you would need and associate those for our functions or services that you want to build. But when you do that you also have to care about how to bring fault tolerance and how to scale the resources.

So, the applications that we would have built we need to configure explicitly to scale, and this configuration is where the management control and flexibility that a developer needs to do. And this, in a way, is good because now you control how to scale, but in a way it is bad because at times, you may want the functions to be running when they want to run in high load, and when there is no load, you do not want to do it.

That means you need some means to do this automatically, but unless you configure they will not be done automatically that is the drawback that we see. But when the other side of it is that if you have scaled and set up these instances, these instances would run forever, and what that means is on a platform you are utilizing these resources to be run, even when there is no load that is really accessing those machines.

So, you are burning basically the idle cycles in those applications. So, you have to pay even when the services are really idle. And whenever you want to bring or boot up a new application, we saw that because you have to run on a given platform, it takes a good amount of time, and this is where the shortcomings that we can see in platform as a service.

And now when we want to do the serverless we want to maintain and integrate the benefits what we see in the platform as a service but take out the challenges that we saw with respect to the the platform as a service. And what that means is we do not want to be having full control or manage the entire deployment environment. We just want to focus on the development logic of the functions and how we want to build them.

And second, when we have that, we also want these applications to be scaled automatically, rather than we as a user trying to configure which can go wrong or may not suit at a certain point, we want to have automation in terms of how they scale up or scale down or scale out or scaling of these resources can happen based on the workload that we see.

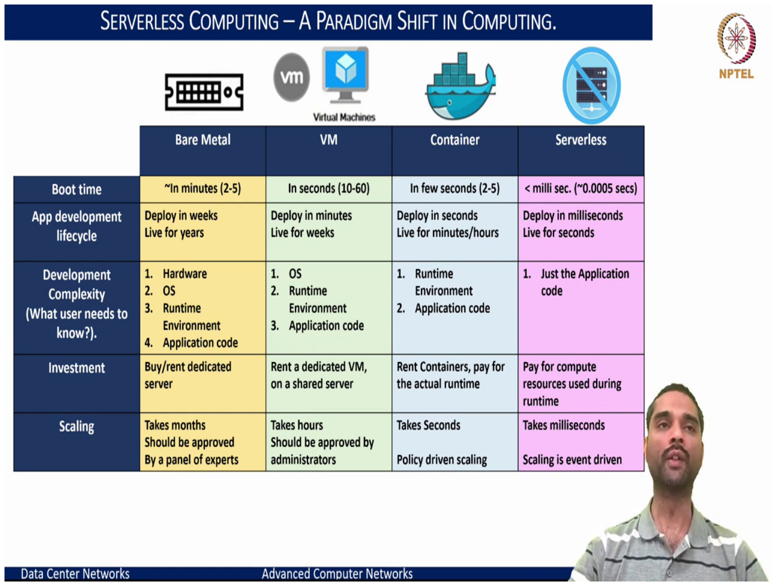
So, if I see that at a given point, there is high traffic and high demand for certain functions to be run, I would want to scale them to meet that request, and at times, if there is no usage, like in the morning, maybe we are trying to watch some news or some articles, which may be on high demand, but as the day fades maybe we are interested in some other kinds of data but not necessarily the same data that you would see.

So, there is a relation in terms of at what time the user would be interested, what users would be interested in what applications. So, if I leave it for the system to automatically scale up when there is a high load and automatically scale down when there is no load that alleviates the burden of trying for us as a developer to manage these resources.

And more importantly, if I have the function and it is not going to be used for a long time I should not be paying for using any of the resources because you are not literally spending any CPU, compute,, or memory resources for running your functions. And only when there is an execution that happens for the functions is when I need to be paying, and this is where again the serverless is in contrast to what the platform as a service provides.

And this application code would only execute when invoked that means you will not be having any compute or memory or aspects to be used when it is not in use, and although you may have some storage where your code may reside, that needs to be brought in and run, but does not mean that you would need a compute infrastructure to run your service. So, you will not be charged for any of these services that you would otherwise have not utilized, and these are the key benefits that serverless facilitates over the platform as a service.

(Refer Slide Time: 32:56)



SERVERLESS COMPUTING – A PARADIGM SHIFT IN COMPUTING.

vm
Virtual Machines
NPTEL

	Bare Metal	VM	Container	Serverless
Boot time	~In minutes (2-5)	In seconds (10-60)	In few seconds (2-5)	< milli sec. (~0.0005 secs)
App development lifecycle	Deploy in weeks Live for years	Deploy in minutes Live for weeks	Deploy in seconds Live for minutes/hours	Deploy in milliseconds Live for seconds
Development Complexity (What user needs to know?).	1. Hardware 2. OS 3. Runtime Environment 4. Application code	1. OS 2. Runtime Environment 3. Application code	1. Runtime Environment 2. Application code	1. Just the Application code
Investment	Buy/rent dedicated server	Rent a dedicated VM, on a shared server	Rent Containers, pay for the actual runtime	Pay for compute resources used during runtime
Scaling	Takes months Should be approved By a panel of experts	Takes hours Should be approved by administrators	Takes Seconds Policy driven scaling	Takes milliseconds Scaling is event driven

Data Center Networks Advanced Computer Networks

So, to sum it up what we have seen is the serverless computing as a paradigm that has been a shift, that has happened from operating with the bare metals all the way to the containers and out of the serverless as a function as a unit of scale, and if we see with respect to the boot times how they vary is you have very few minutes to receive power on your machine, it takes around a few minutes to boot up.

While the VM you would see that it takes in few seconds to boot up, while the containers being lightweight, they would boot up in few less than 10 seconds, but when it comes to serverless, you are talking about the functions alone, that means you click, and the function executes, and that is less than milliseconds to microseconds time.

Although sometimes the times very vary based on what kind of runtime requirements that they have, but you can see that it is much lower in the orders of seconds as opposed to any of the earlier models. And with respect to the application development life cycle, you can deploy those in milliseconds and run them just for the time that you need, rather than making them to persist and be there even when they are not going to be utilized.

And all this in terms of management aspects, if you see, the complexity for a developer to think of, because if I am writing a code and that is going to run on a VM and I have to care about the OS, I have to care about the runtime, the environment and then the application code as well, but when it comes to containers, you do not care as much about the OS, but just the runtime in terms of what kind of a container framework that you are using and then run it.

But when it comes to the serverless, you are just speaking about the application code, and that means it is a lot more easier and better for a developer to focus on just the application logic and build the code. And this also means that in terms of usage, it is going to be a lot more simpler and in terms of the money, both in terms of the capital and operational expenses, it would be much more lighter as opposed to any of the other models.

Because if we have to talk about the bare metal, we have to purchase the hardware or at least rent the hardware from some dedicated vendor and then deploy the OS purchase, whatever the necessary operating system licenses, libraries, and so on, and then build. So, there is a lot of capital expenses as well as when we have these devices, we need to manage the routine updates checks and handle all of this, so there is also going to be operational expenses.

But when it comes to the VMs, because you rent a dedicated VM on a shared server, your costs are somewhat going to be marginalized. And likewise, with respect to the containers where you are going to rent specific containers and not dedicated VMs or the CPU or a compute requirements per se, this minimizes further in terms of the capital expenses that you would do and also items of the operational expenses.

And when it comes to the serverless it completely eliminates the need for any capital investment and for the functions that you develop you would pay just for the time that you would really utilize the compute cycle. So, it is also bringing down a lot of expenses and in terms of the resources that you would not want to spend on.

And likewise with respect to the scaling of these resources, so if I have one machine, I had have to stack up another machine, to scale it takes definitely months of times, to procure set up, the entire infrastructure, replicate the entire software stack and then build it and it also requires a lot more expertise.

And when it comes to VMs, you still need specific administrative aspects to be handed, but it is a lot more easier than looking for bare metals, and we can scale it with just a click on our site to say I can replicate this VM, provided we have sufficient resources, we could spawn a new VM and run it. And likewise with respect to containers, but you need to still manage when I scale these containers, how each of these containers are going to be used, what is the policy that dictates how to scale, all of these aspects need to be defined by the developers.

But when it comes to the serverless on the very other extreme end you leave it to the platform to manage this scaling and we can see that it takes a lot less time to boot up and run the services. It also means we can take these scaling decisions in a very small time scale and this is what we essentially refer to as scaling when event-driven my system where the things can be done in a much more quicker manner.