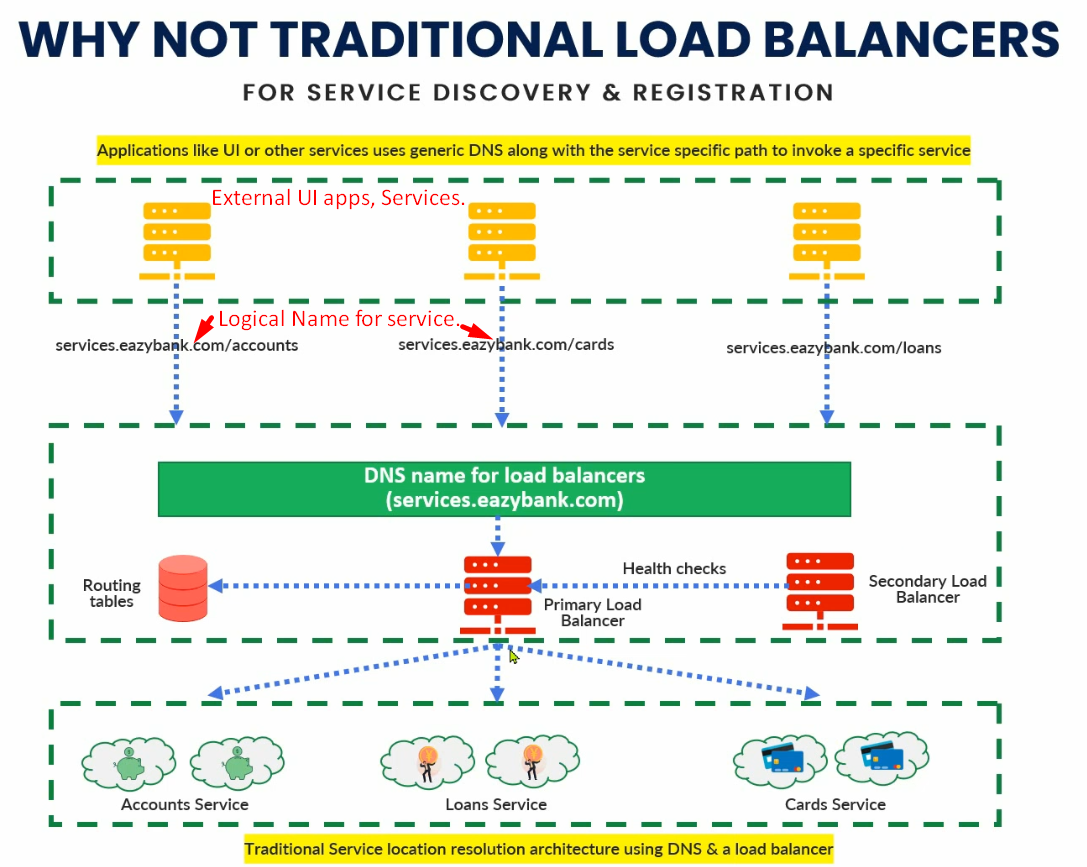
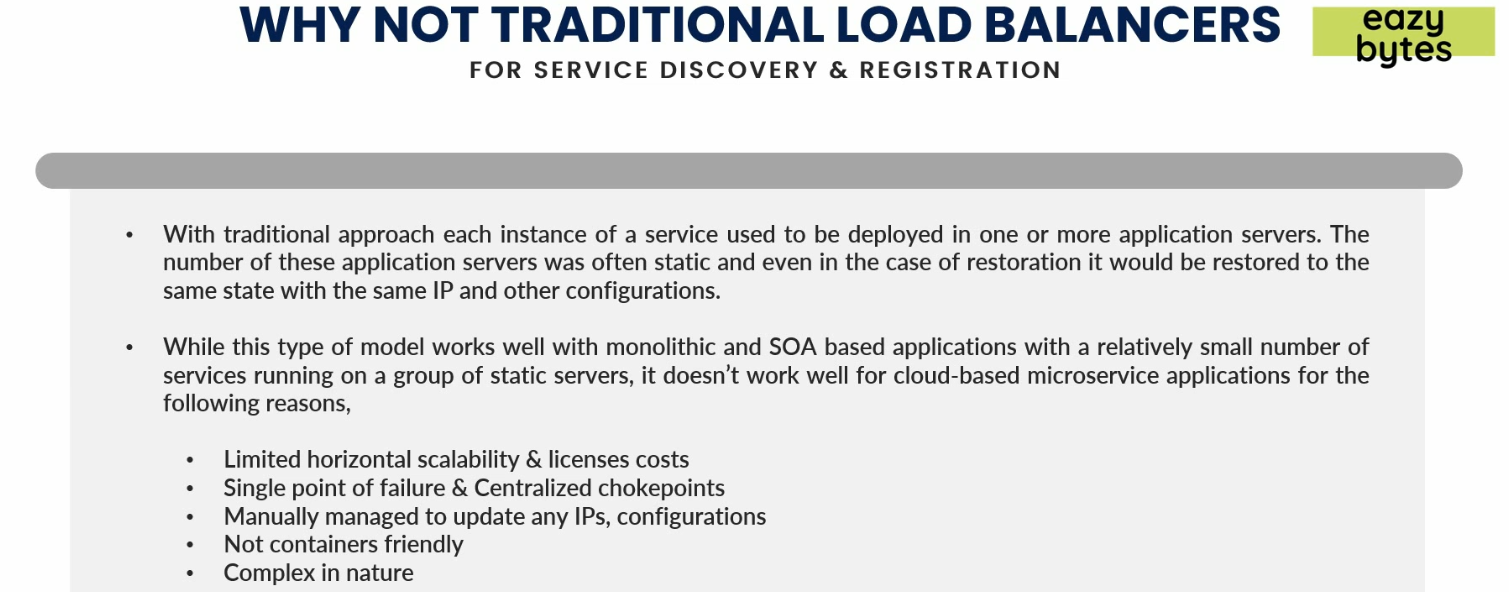
1. **Agenda**:
   1. Why we can’t use **traditional load balancer** in our microservice architecture?
2. If you don’t know how a load balancer works, we will study in this lecture.
3. See the architecture, inside an app, there is a load balancer.
4. 
   1. If you observe the architecture diagram where inside an app, there is load balancer in b/w.
   2. Apps like UI or any other service that wants to interact with my service, they can call my services with logical Service names for our microservices are exposed to outside world.  
      Such as services.eazybank.com/accounts for accounts microservice.
   3. UI Apps or other apps call our microservices using these logical names.
   4. Based on these logical names, inside load balancer, we do **DNS name mapping** 🡺 **Domain Name System**  
      Like for this service, this is the IP address and port no that is stored in **“Routing Table”**
   5. So, whenever someone invokes load balancer with logical name, load balancer goes and checks the **routing table** for all the instances and their IP addresses for that particular logical name.
      1. **For example**: If we invoke services.eazybank.com/loans, the Load Balancer knows there are two instances of this services currently running from the Routing Table with this and this Ips and Port numbers.
   6. Secondly, Load Balancer has internally its **own logic like Round Robin** way or **geographical location** way of balancing the requests among the different instances of a microservice. So based on that particular logic, Load Balancer will make a call to a particular instance.
   7. We have a primary and Secondary Load Balancers.
   8. Primary load balancer is centralized load balancer which means all the requests land on this load balancer so it has great dependency.  
      If it is not working, no request can be served.  
      So, in traditional load balancer, a secondary load balancer is maintained and is always in standby mode.
   9. The secondary load balancer keeps on sending “health check” to the primary load balancer like “Are you doing fine?”
   10. If the primary’s health is not ok, immediately secondary load balancer takes the responsibility and starts serving the requests.
   11. So, in this way, the traditional load balancer will make sure that the requests are being processed through a logical service name rather than directly invoking through IP numbers and port number.
   12. If we expose IP and Port to your customers or UI apps who want to consume your backend services, there are some problems
       1. Which is a starting point for hackers to hack your app as they can find out IP address through JavaScript Code.
       2. If you change IP (server), port number, you need to convey these changes to all your clients which is not an ideal for enterprise solution.  
          **NOTE**: For this, we have load balancer which takes care of them in our enterprise app.
5. Now coming to the question
   1. 🡺 Why we can’t use “Traditional Load Balancer” in our microservice architecture?
6. There are multiple issues if we use “Traditional Load Balancer” in our microservice architecture.  
   
7. Reasons:
   1. Let’s first understand why this will for monolithic and SOA based applications.  
      Monolithic or SOA apps are deployed on one or two servers and very rarely we change server so they are static in nature.  
      If you are going to restore a server, it will be restored to the same IP address. There is no need of dynamic change of IP address and port no in traditional app because there are limited no of services/apps to be taken care of.
   2. This will not work for cloud based microservice app for the following reasons.
      1. **Horizontal scaling of primary load balancer is costly and tuff**: If you have 100 microservices, the kind of load you can imagine that load balancer will have to handle. If you think about horizontal scaling for this problem, then we have limited options to horizontal scale up traditional load balancer.  
         If I want to create a cluster for primary load balancer, it is very tuff and involves a lot of cost.
      2. **Single Point of Failure**: As we have only primary and secondary, what if they stop working and at the same time, we are choking our centralized location with a flood of requests.
      3. **Manually configurations for IP addresses**: Whenever a new instance is created, it needs to be registered manually.  
         In microservice architecture, we have 100 microservices and we are scaling up or down frequently and that too dynamically.
      4. **Complex in nature:**
         1. This is not container friendly whereas we deploy our microservice inside container as a cloud native app using docker image. But traditional load balancer works well for big server.

The architecture of traditional load balancer is very complex to set up and a good team is required to monitor it on day to day basis.